

Cardiff University

School Of Computer Science & Informatics

INFORMATION EXTRACTION FROM WEBPAGES TO INFER NEW KNOWLEDGE USING ONTOLOGICAL MODELLING.

Author: Ashley Sean James

Project Supervisor: Dr. Andrew Jones

Academic Year: 2015-2016

Submitted in support of the degree of
Computer Science B.Sc.

Abstract

With the coming of the third generation of pervasive and ubiquitous computing, the amount of information which is being stored on internet infrastructure is continuing to increase. Information about almost any topic is available, yet web page providers are actively controlling the content we require. As such, evaluating results from even reputable sources can be both costly and time consuming.

The work carried out as part of this research project investigated the feasibility of automatically extracting information from a range of websites to infer new knowledge, allowing more informed decisions to be made more quickly. However, this knowledge extraction and fusion task is difficult because information must be retrieved effectively and modelled appropriately based on the nature of the extracted information. An intelligent information model could allow a programmatic system to start to make automated judgements based on the information obtained.

In summary, this study and its associated concept prototype have shown that “information extraction” from website content to infer new information is indeed achievable. Furthermore, the principles of the developed solution are generic across information domains which are available for knowledge inference. An Artificial Intelligence agent of this calibre can infer new knowledge and learn from the websites it visits through creating a model and set of properties for a given subject matter.

Further work could lead to the design of a commercial solution which would be a major step toward an era of smarter computing, based on reasoning, inference and assertion rather than data analysis and “*Big Data*” based solutions.

Table of Contents

Executive Summary	4
Glossary of Terminology	5
Section 1 – Background & Introduction	7
Section 1.1 - Background	7
Section 1.2 - Introduction	8
Section 1.3 - Aims & Objectives	8
Section 1.4 - Anticipated Outcomes	9
Section 1.5 - Principles & Approach.....	10
Section 2 – Research Approach & Findings	11
Section 2.1 - Technologies & Techniques for Information Extraction	11
Section 2.2 - Building the Ontology to Infer New Knowledge	11
Section 2.3 - Additional Research: Building a Hierarchical Food Chain	12
Section 2.4 - Implementation Technologies Available.....	12
Section 2.5 – Anticipated Challenges.....	14
Section 3 – Development Method.....	16
Section 3.1 - Development Summary	16
Section 3.2 - Development Approach	17
Section 4 – Problem Specification & Design.....	20
Section 4.1 - Problem Specification	20
Section 4.2 - Design.....	23
Section 5 – System Implementation & Class Descriptions	27
Section 5.1 - System Implementation.....	27
Section 5.2 - Detailed Class Descriptions	36
Section 6 – Testing, Results and Findings.....	43
Section 6.1 - Testing.....	43
Section 6.2 - Results and Findings.....	43
Section 7 – Future Work And Personal Reflections	47
Section 7.1 - Future Work	47
Section 7.2 – Personal Reflections	48
Section 8 – Conclusions	50
Appendices	51
References	96

Executive Summary

This research project proved that *“Information Extraction to Infer New Knowledge Using Ontological Modelling”* is feasible. This was demonstrated through a concept prototype which was developed using iterative software development techniques. This investigative prototype used reasoning, inference and assertion over traditional data analysis and Big Data based solutions. By extending the core principles further, a future collaboration between industry and academia could lead to the design of a commercially applicable knowledge generation solution. This could bring more organisations into an era of smarter computing.

To help validate the principles, the topic of “animal classification” was chosen to research into and explore the prototype’s capabilities. Initial phases of prototype development focused on extracting information from the web, based on the classification of given animals. “Jsoup” was selected for “information extraction” tasks, because it offered much of the functionality needed to retrieve HTML information from a given web-page. The initial challenge being to extract the classification of a given animal (e.g. “Bird”, “Reptile” ...).

Developing this approach, a “voting system” algorithm was designed so that results from several web-sites could be combined to yield the most likely classification. Using JENA, an animal properties ontology was then constructed. The determined classification made it easy to visualise how a given animal should be classified with its corresponding properties.

Further development effort focused on “inference” using a reasoner, to assert what *“animal classification”* a given animal belonged to. e.g. if an animal such as an “ostrich” “has” the property of “wings” and an *“animal classification”* “Bird” is “equivalent” to animals with “wings”, then an ostrich can be inferred to be a “Bird”. Other useful properties were captured e.g. an animal’s scientific name; proving that adding extracted properties was achievable.

Additional implemented improvements included using the “Levenshtein distance” to calculate the similarity of two strings. Specifically for this project, the similarity of the input to a specified set of “acceptable” *“animal classifications”*. This meant that even if the system was to extract a classification containing a “spelling-error” then the system could standardise this incorrect input. e.g. {“Snake”, “eptile”} could be corrected to {“Snake”, “Reptile”}. Other standardisations included cases of technical and biological terms (such as “Aves”=“Birds”).

Constructing a dictionary of these “other cases” meant that the value of an entry could easily be standardised to its “key” and would result in a more appropriate classification. Functionality for human intervention was included when an animal could not be classified. Such modifications were easy to implement because of the extensibility of the solution.

Final development iterations focused on the novel concept of building a “Food Chain” hierarchy of animals and their predators and prey. A recursive function was designed to search through the possibility tree based on what an animal “eats” and is “eatenBy”. These relationships could then be added to the model and inverse relationships could be inferred, building up a more complete animal ontology. This feature could be used as a very powerful classification tool by biologists e.g. using the recursive algorithm to retrieve the classification and properties for each of the animals considered during the traversal process.

Glossary of Terminology

Technical Term	Definition
Agile Development	A process for developing software in an iterative and incremental manner.
Animal Classifications	Classifications which animals can be put into. (E.g. Mammals, Birds, Reptiles, Fish, Amphibians).
Animal Properties	Features of an animal which make them unique and belong to a certain animal classification family. (e.g wings, scales, gills).
Big Data	Huge quantities of often complex data which require significant processing to analyse.
Eclipse IDE	A Java software development programming environment supported by the Eclipse Foundation using tools to handle “plug-in” dependencies.
GitHub	A web based repository hosting service providing benefits such as backed up and versioned code.
GUI	Graphical User Interface
IDE	An Integrated Development Environment is a software application which provides programming specific facilities for software development.
Information Extraction	The process of extracting information from webpages
JENA	A Java library used to build the ontology model, containing methods to create and add properties, classes and reasoners to a model.
JENA Ontology Objects	JENA (com.hp.hpl.jena) specific implementations for Classes, Properties and different flavours of Model superclass. e.g. OntClass, ObjectProperty, OntModel, InfModel.
Jsoup	A Java library which contains functionality to perform information extraction tasks, such as retrieving HTML and searching for patterns within HTML.
Maven	A dependency management tool which is implemented in Eclipse.
NLP	Natural Language Processing – applying computational models to textual formatted data.
Ontology	Hierarchy of related concepts which are often linked by properties.
Ontology Creation	The process of modelling an ontology of concepts programmatically using a utility.
Org.apache.Log4j logger	Logger for org.apache libraries.
OWL	A web ontology language which is a set of markup languages which are designed for machine processing of information.
Pellet	A reasoner which can be used to perform complex reasoning tasks.
Protégé	A graphical tool from Stanford University used to build up Ontology Models.
RDF	Resource Description Framework - a framework for describing resources on the web.

RDF triple	Consists of a predicate, subject and object. E.g A cat has claws. Predicate: "Cat", Subject: "has", Object: "Claws".
Reasoner	A reasoner uses a set of rules to categorise different classes based on their properties.
Regression Testing	A code module is initially tested and future testing includes the previous tests to check code changes have not affected program correctness.
StringUtils	A complex Java library for handling strings.
Turtle	Terse RDF Triple Language - a way to express data in RDF triples in a simple format.
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

Section 1 – Background & Introduction

Historically, “information extraction” has been an interesting research topic and continues to be an area for profitable research. The initial prototype solution implemented for this project, began as a set of ideas on how to manage “information extraction” processes more efficiently. It later developed into a comprehensive “information extraction” and analysis engine.

Section 1.1 - Background

Information identification, extraction and analysis has always been an important area for commercial and academic research. From analysing ancient hieroglyphics to extracting information embedded within internet based web-pages, this is still the case. Businesses need to be innovative and quickly react to changing market trends and influences. It is through extracting high-quality intelligence of this type that the most profitable decisions can be realised.

The nature of the internet is content based and is always developing and evolving. New content is constantly being added and this makes the challenge of successful and accurate “information extraction” ever more difficult. Our dependence on information was well expressed by Geoffrey Moore at the Hadoop 2012 Summit: *“Without big data you are blind and deaf in the middle of the highway”*. We are surrounded by data and it is integral to the way businesses are being run. Discovering data trends and reacting to them, is similar to being able to see and hear.

Companies are looking to analyse website content and research has focused on the applications of “*big data*” technologies and approaches. The idea is to take lots of connected (and disconnected) relevant data and bring it all together to allow information to be assessed for decision making purposes e.g. looking for interesting trends and key pieces of information which can help enhance a company's profitability and market appeal.

In 2010, Google's CEO Eric Schmidt summed up the sheer volume of information being generated: *“Every two days now we create as much information as we did from the dawn of civilization up until 2003.”* If this huge amount of information is processed correctly and turned into useful and applicable knowledge, then the advantages and benefits are clear.

Many different sources can provide useful information and combining these sources is a challenging task. However, combination approaches can often lead to better information quality and may even lead to the discovery of previously unknown facts. Checking that the processed information is suitable either involves additional processing or the use of cross-referencing capabilities.

Automating such “search and learn” processes would be very productive from a commercial perspective. However, the problem lies with the computer's ability to build and classify objects in such business models. This is usually achieved using Natural Language Processing (NLP) which is a technique for searching for nouns, verbs and adjectives and using these as the basis to make up triple (subject, predicate and object) statements.

One of the key reasons why information extraction is so difficult is because “*Information may need to be combined across several sentences...*” [and] “*There are many ways of expressing the same fact...*” (GATE Information Extraction, The University of Sheffield). These problems were challenging tasks to be overcome during this project.

Section 1.2 - Introduction

Techniques involving “big data” are commonplace in industry and the push to analyse and collate the data we have on the internet to infer new information is an area which still requires further investigation. Having considered the available techniques, it was decided to explore how easily internet content could be extracted, structured and built up into an ontological model to help infer further knowledge about a domain area.

Taking useful and applicable information out of a piece of text is a simple task for most humans. However, for a computer based system, this is quite a complex task. Humans use characteristics and know what sort of information is being looked for almost intuitively. Rule based reasoning to infer new information in this way is a lot more difficult for a machine to perform. Most of the time, a human would be able to make this generalisation without thinking. The machine does not know that there are cases where rules need to be taken in a more “fuzzy” or relaxed manner.

“Information extraction” as a process can be achieved through retrieving a website’s HTML page structure and then looking for an associated pattern. This pattern can then in turn be used to locate specified phrases from within the text of the HTML content. Through using an extraction tool this HTML content can then be parsed and useful information determined.

“*Animal classification*” was the domain area chosen for the prototype system and it is an interesting study topic for biologists, who aim to classify generations of animals based on their biological attributes. To explore these possibilities, animals were classified based on their properties determined from several web-sites to ensure accuracy. N.B. There are many challenges which complicate the modelling task e.g. issues relating to the way natural languages are formed or the suitability of the websites where information is extracted from. Currently, no such automated system to perform biological inference exists and biologists still classify animals based on their own knowledge.

The benefit of the developed prototype system is that it is extensible to any information domain with suitable configuration and coding adjustments. The core algorithms are appropriate for extracting information for a whole range of different considerations e.g. through studying information about prehistoric animals, the algorithm might facilitate new animal features and allow scientists to better reason why extinction occurred (e.g. due to a lack of suitable biological attributes under certain environmental conditions).

Section 1.3 - Aims & Objectives

One of the project’s key aims was to achieve “information extraction” from webpages to infer new knowledge, using ontological modelling techniques. The theme of “*animal classification*” was an appropriate choice because animals are a readily available knowledge resource online on “A-Z animal” websites.

Using an animal's properties (e.g. "wings", "scales" etc.) the associated "*animal classification*" (e.g. "Reptile", "Bird") was determined.

An important aspect of this project was to determine whether ontological modelling was feasible and could be successfully implemented and demonstrated using a prototype. A further challenge considered was whether the system could adapt to more difficult classification tasks, such as "a mammal" having "wings"? e.g. a bat.

Additionally, the idea of creating a food chain of animals based on the properties of "eats" and "eatenBy" was investigated and implemented. Given one "starting" animal, it was found to be possible to quickly build up such a food chain using recursive algorithmic techniques. This proved to be a very interesting and novel part for inclusion in this dissertation.

Section 1.4 - Anticipated Outcomes

The following goals were established:

- Extracting information based on known patterns found on webpages.
- Searching webpages for specified phrases to aid in classification.
- Use information extraction techniques to model extracted data appropriately.
 - Using modelled information, inference processes can determine what classification an animal belongs to.
- Add suitable extracted information (which is in subject, predicate and object form) as triple statements to the ontology.
 - Additional information such as a domain can be added to the ontology.
- Investigate whether a food chain hierarchy of predators and prey can be constructed.
 - "Infinite loops" may occur in the recursive solution if a termination condition is not correctly specified.
- Look at techniques for achieving better quality extraction results.
 - e.g. using a "voting" system which calculates the most likely result.
- Produce a robust solution, adaptable to many different webpages (whether information is extracted or not).
- Standardise results because websites may have correct information but for a different "animal name".
 - e.g. The Latin name "Panthera Leo" also means "Lion".
 - Fish is a high level animal class; many websites use more descriptive lower level subclasses. E.g. Actinopterygii – ray finned Fishes.

Section 1.5 - Principles & Approach

A summary is provided below of the key principles of the prototype system, as well as the high-level approach taken. *Section 3* is devoted to the development method and system scope is discussed in further detail in *Section 4.1.1*.

The core principles were:

- An ontology will be used for modelling and reasoning purposes.
- Only animal websites will be considered for classification.
 - e.g. “plant” related websites were not considered.
- The main classification groups considered are “Mammal”, “Birds”, “Reptiles”, “Amphibians” and “Fish”.
 - “*Animal classifications*” such as “Invertebrates” were considered out of scope and the diagram below shows what was included:

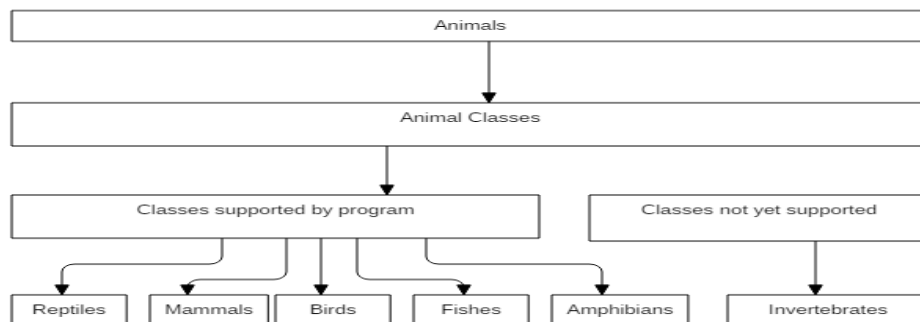


Figure 1: Hierarchy of Supported Animal Classes

- All animals within a given main classification group will have a specified defining property which makes them belong to that “*animal classification*”.
 - If an easily defined animal can be classified, then with a similar approach and more precise investigation, it is possible to define more complex “edge-cases”.
 - The system classifies animals with defining properties (e.g. Bird: “wings”).
 - For difficult edge-cases (e.g. “Bat”) the system was meant to classify these, although this was not crucial.
- When adding animals to the “Food Chain”, only animals which were included in the given “A-Z” list of accepted animals would be added.
 - This ensured results which included “plants” or “non-animals” were ignored.
- Focus on building the mechanisms to achieve “information extraction” and not the interface for the solution.
- The approach taken was an iterative development method which implemented and developed functionality across a number of iterations.

Section 2 – Research Approach & Findings

Research activities were partitioned into two important areas: (i) information extraction, and (ii) ontology modelling, and were performed concurrently as an efficient learning style. This required more insight but was worthwhile due to the potential for greater understanding it afforded. When good progress had been made with the main development tasks, it was possible to research into a “Food chain” of predators and prey, documented below.

Section 2.1 - Technologies & Techniques for Information Extraction

Looking into the primary research area allowed me to better understand some of the best techniques and technologies available to extract information from web sources. As part of this, the most useful web sites were explored for extracting information. Looking into web page structures allowed decisions to be made about web-page suitability. It was found that many research papers were not relevant to what was required. e.g. “building trees” and “wrapper induction” techniques (Sigletos.G et al).

An early decision was that these more complex techniques were not required for this project. The challenge was to take a simpler “information extraction” tool (which had already been implemented and documented) and use the webpage information it obtained to infer new knowledge. This simple approach to extracting core information could then be expanded in future development iterations to use more sophisticated algorithms. The primary ambition was to do something novel and not just follow previous approaches.

Libraries were looked for which could integrate with Java because they would combine well with my knowledge of class based programming and the extraction functionality already implemented. Jsoup was identified for this as it contained the required methods for “information extraction”.

Section 2.2 - Building the Ontology to Infer New Knowledge

Primary investigations focused on building up the ontology (creating properties and relationships) to reason with the different ontology classes to infer new information. Starting from the very basics of understanding the ontology modelling goals through to implementing small yet efficient test prototypes.

The Protégé ontology specific modelling tool was used to test the information models and understand how to infer information. Protégé allowed relationships and properties to be expressed in an OWL format. Protégé provided a very useful “sandbox” environment for researching into the way the ontology modelling process worked.

Additionally, Protégé provided many benefits including being able to visualise what were the properties and what needed to be added, to being able to see what the generated “turtle” file OWL code should look like. Functions were designed to achieve this similar look and feel.

Much was discovered about how ontology properties can be added and different types available e.g. “transitive” and “inverseOf” property relations.

All this acquired knowledge helped to build an understanding of what was required to build such a comprehensive ontology. The goal was to emulate Protégé OWL code using Java.

To simplify the “ontology creation” process, a set of functions were designed to handle more complex functionality. Further investigations and testing activities identified JENA as a suitable choice and was perceived as a familiar technology. It featured all the functionality required and was relatively easy to add to the project. Maven was used to handle JENA dependencies and the Eclipse IDE supported this.

Section 2.3 - Additional Research: Building a Hierarchical Food Chain

This was an additional component which was designed and implemented once the core prototype solution was implemented. This functionality was mainly designed from the preceding research and through applying knowledge of recursive routines it was possible to create an ontological model which could support a large hierarchy of predators and prey.

“InformationExtraction.java” was used to extract the information for a given Predator or Prey, using the method designed for pattern extraction. Likewise a new method was created for “OntologyCreation.java” called “addTriple” which added a property to the model based on two OntClass objects and a given ObjectProperty.

The challenge was getting the process to prevent infinite looping due to recursion. Choosing a suitable termination condition to end the recursive process required an understanding of the core algorithms underpinning the solution. Without such a condition, an infinite looping situation could arise where one animal “eats” another animal, but that secondary animal also “eats” the first.

Section 2.4 - Implementation Technologies Available

Ideally, a complete and packaged solution was needed to achieve modularity and extensibility i.e. one did not want to perform “information extraction” with Python and “ontology creation” with Java. A packaged solution would allow the development of a more deployable solution. These key decisions were made based on whether the programming language was modular and well documented, as was specified by *Requirement D7* (further detail about all the requirements can be found in Appendix A).

Taking into account programming language needs extracted from the requirements backlog, it was concluded that either Java or Python could be a suitable choice. The industry standard of GitHub was used to maintain a project repository to backup and version code.

The web-sites explored by the prototype were:

Web-Site URL	Information Extracted
http://kids.sandiegozoo.org/animals/	Animal classification and properties.
http://a-z-animals.com/animals/	Animal classification, predators and prey, scientific names and properties.
https://en.wikipedia.org/wiki/	Animal classification and properties.

Of the animal websites investigated for their suitability, the three listed above were deemed the most useful for their content and testability aspects. Appendix F lists the criteria which were used to select suitable websites and descriptions to detail why each site was selected.

An overview has been provided to specify the components which were required and how well Java suited these needs.

Components For A Java Approach

Java was selected because it is modular, well-documented and allowed for future extensions. Its class based approach suited the proposed solution. The solution packaged up better in Java and dependencies could be resolved using Maven.

It was appreciated that the key functionalities required for “information extraction” and building the ontology, would have been previously implemented in such a supported and common language of choice. Overall the process of adding a “reasoner” was straight forward in Java and would be a lot more extensible than Python.

Eclipse IDE

The Eclipse IDE was used to integrate the solution because it had the functionality to maintain and resolve dependencies using Maven. It was also possible to add JAR files and additional, more specific functionality as required. Such functionality was required for both Jsoup and JENA, the main Java libraries chosen for this project. Eclipse also “auto-completes” imports for the Jsoup and JENA code which has been entered.

As an IDE solution it offers code completion and error underlining notifications, as well as simple, template method stubs. It is well documented online and supports “debugging” and program execution.

Jsoup

Jsoup is a Java library which contains many useful methods for “InformationExtraction” required for the solution. It has methods to get the HTML contained in a web-page and also search through its contents using a regular expression like syntax for certain “patterns”.

Jsoup also implements “AND” and “OR” functionality for finding specific patterns, which is very useful when looking for several different words. Jsoup is a well-supported and documented library and is quick and easy to use due to its sensible naming conventions. Overall, these features of Jsoup made it very suitable to be used to extract information.

The following primary Jsoup methods were used:

- Jsoup.connect(String URL).get() : returns Document
 - Function to get the HTML code of a given URL and return the data in a Document format.
- .select(String pattern) : returns Elements
 - Looks for HTML which follows the “patterns” specific format, whether the pattern contains HTML but also regex statements.

- This is useful because other functions such as “:contains() and :matches() can be used to find more specific words inside of a select statement, with the syntax of “: select(:matches(text))”.
- :contains(text) - Will return any HTML in the form of an Elements object which contains the specified text.
- :matches(text)-Will return HTML which exactly matches the given text.

JENA

JENA is an excellent tool to build up an ontology; it allows a model to be built and properties to be added to classes. The ontology model can then be queried using a reasoner. It also offers functionality for printing a model.

Some of the JENA functions used:

- animalModel.createClass(ns + aclass) : returns OntClass
 - OntClass created for a name space and aclass.
- animalModel.createProperty(ns + property) : returns ObjectProperty
 - ObjectProperty created for a name space and aclass.
- animalModel.addSuperClass(aClass): void
 - Adds a superclass relationship between two classes, sometimes using the someValuesFromRestriction() method associated with a model.
- ModelFactory.createInfModel(Reasoner pellet, Model oldModel) : returns InfModel
 - InfModel created applying a reasoner to a specified oldModel.

Components For A Python Approach

Python offered the BeautifulSoup library as its “information extraction” tool, which was an appealing choice because of its simplicity. BeautifulSoup is the Python library equivalent to Jsoup and contains much of the functionality which the Java equivalent implements.

Python also offered the facility of “Seth-scripting” for OWL Ontology Building. Seth is a Framework designed to integrate Python with the OWL technological approach. It has a very simple syntax and is a good Python equivalent to JENA.

Python was rejected because it was lacking the more complex functionality required and even though it contained some of what would be required for the project. Python is used less often in industry.

Section 2.5 – Anticipated Challenges

After researching into these key areas for a Java implementation, the following challenges were identified:

- JENA is poorly documented with few code exemplars to study. Mitigation would require research into each specific challenge to better understand what was required.
 - Protégé was used to test that correct results were being obtained and then JENA libraries were used to build up suitable classes. These classes were then used to simulate OWL syntax properties, classes and assertions.
- Several additional functions were required to handle variant circumstances such as mammals being specified as “mammalia” etc.
- Extracting information from unseen pages would be difficult.
 - The approach is dependant on the content stored within the page; if this contains a pattern that has not been coded previously then alternate pages will need to be used to obtain classifications.
- Some web-pages may not feature the required information, which is why analysing a set of web-pages is necessary.
 - A function to count the number of times a given extracted word occurs in a list would be needed.
- Error messages would need to be quite specific, if Jsoup cannot return the correct result.
 - Both of the following functions would require exception handling:
 - Getting HTML from a page.
 - Getting the properties for a given animal.
- The program would not work effectively for every single animal website, as it is very difficult to perform “information extraction” for all pages.
 - Strange or unexpected pattern layouts would be difficult to process.
- Problems such as spaces in words would need to be handled appropriately.
 - When searching a given URL (a base URL and appending a given animal) care would be needed with spaces e.g. adding a special character so that the URL is correct. For example, “wild boar” -> “wild_boar”.
- Implementing the “food chain” capability would be a more difficult programming task due to the recursive nature of the search algorithm.
- The tight project timescales would require good planning and efficient working.
 - An iterative development approach would allow the most important functions to be implemented first.

Section 3 – Development Method

The development method featured an agile methodology, making use of iterative prototypes and feedback to facilitate prioritised change. This section summarises this approach, how it was implemented and the development practices adopted.

Section 3.1 - Development Summary

An agile development approach builds up a system through multiple development iterations, focusing on implementing requirements based on their priority. Requirements were maintained on a prioritised product backlog with the most important requirements stacked on top and applied to the next iteration (sprint).

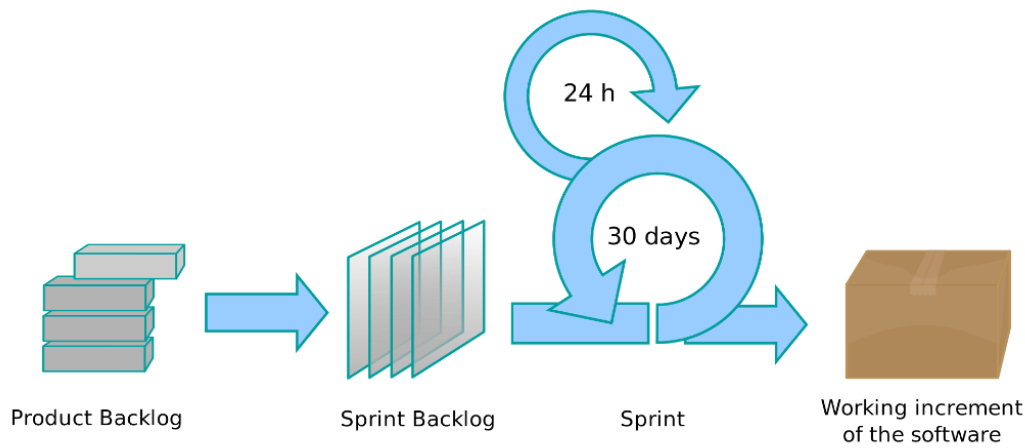


Figure 2. Agile Development Overview

Through building up a deeper understanding of the requirements during each iteration, it was possible to become more proficient when coding in later iterations and with less re-work. It was decided to mitigate risk through focusing on researching and implementing the high-value, lower risk functional elements of the system. This meant that some low risk components were implemented in earlier iterations to provide a greater overall understanding of how to combat the more difficult requirements.

This was a different approach to the traditional Linear Waterfall model initially considered and was a decision taken to allow for complex and novel features to be researched further and implemented incrementally. The complexity of the subject matter meant that it might not be feasible to implement the entire system as part of a single development cycle. The iterative approach therefore allowed for several complex features to be designed and integrated separately but able to work together to produce the required interesting and novel results.

A primary ambition was to produce a project which would provide further insight into an interesting research area. The initial project proposal was focused on “information extraction” to allow an investigation into how easily information could be extracted from many different sources and forms of website content. This would require in depth research into “information extraction” techniques.

A further objective was to construct an ontology for modelling and subsequent reasoning purposes, although this was not a top priority for the first iteration. As the project developed, it became clear that it was more beneficial to focus on how easily “information extraction” could be used to infer new knowledge. This is a relatively new area of research and little prior research effort has explored and documented the principles this dissertation has covered.

The basic project approach was to research a specific feature and its requirements. This feature could then be designed and what was needed for that requirement could be visualised using a Unified Modelling Language (UML) class diagram. The “StarUML” tool was used to build the necessary design models and was used to document the system architecture through both class and sequence diagrams. This technique was able to highlight where more specific research was needed to improve the solution.

Once all required research information was obtained and understood, effort was focused on implementing a simple prototype in a separate project structure to make testing and coding more efficient. After some adjustments, testing that a feature was performing as expected, under a number of different input conditions, the method was integrated into the evolving prototype.

This latter step (involving further integration testing as the program became more complicated) took longer to complete. Taking what had been learnt from the previous iteration, it was possible to examine the requirements backlog and repeat this iterative process for the next batch of the most important features. Occasionally, it was necessary to introduce some extra features to complement the iteration requirements, which were then updated along with associated designs and relevant algorithms.

In general, end of iteration prototype demonstration sessions allowed feedback to be gathered on progress and reprioritisation of the requirements backlog.

Section 3.2 - Development Approach

The first step taken for this project was to look into existing research in the two key domain areas of “information extraction” and “ontology creation”. Current ideas and practices for “information extraction” were reviewed for their applicability. When looking at the “ontology creation” domain, it was noticed that there was a lack of research into the use of ontologies to model information extracted from the web. It was then decided that this would be a very interesting research topic to investigate further.

When discussing my proposal and “ontology creation” ideas with my supervisor Dr. Andrew Jones, it became apparent that the solution had several useful applications in the area of Bioscience and overall in classifying biological phenomena. It was then agreed that “*animal classification*” would be a good domain choice, because it is an easily accessible resource and as a biological phenomenon, animals presented an interesting topic for further research.

With this more definitive proposal, research work continued but with a focus on how to model the extracted information more effectively. “Protégé” was used to model different reasoning scenarios initially and at a very basic level; modelling got more complicated, once greater experience and understanding was obtained.

Protégé was an integral resource to examine the elements which made up a given scenario, as well as providing insight into the way the “reasoner” was performing its inference. Protégé also provided a way to look at the generated OWL Turtle code for a scenario. This allowed one to then look into the different programming technologies which were available and which could achieve similar ontology building and reasoning capabilities.

After this research into Protégé it was found that ontology properties (such as “inverseOf” and “transitive”) were a powerful way to perform the hard work in a reasoner based model. One of the programming scenarios was to construct a “Food chain” of predators and their prey and investigations began into how this could be achieved in Protégé. It transpired that the best way to achieve the main functionality of classifying animals was to use a reasoner to perform inference.

An animal class would have a set of defining properties and a given animal would also have a set of specific properties; based on these two property sets partly matching (e.g. an animal would need to have all the defining properties which made it an animal of that class) that would then allow an animal to be classified by the reasoner.

Prior research into the area of “information extraction” was reviewed. In particular, identifying the best ways to extract information from HTML content, specifically looking for words or phrases. Looking at the core ideas needed to support the solution, it was decided to use a Java library which supported regular expressions which would be needed in later iterations to match word patterns. A number of different “information extraction” tools were investigated, but most were inadequate. This was because they were too complicated (e.g. Stalker) or did not possess all of the required functionality.

In terms of “ontology creation”, the theory of ontologies, their properties and reasoning techniques were investigated further using Protégé, as well as looking at the programming libraries which were available to achieve this functionality and which Protégé used.

Further investigation and reading articles on the two key areas of “information extraction” and ontologies, revealed that two of the best tools to use for this project were the Jsoup and JENA Java libraries. This was because they could be seamlessly integrated with an Integrated Development Environment (IDE) such as Eclipse. Most importantly, Jsoup and JENA contained the key methods required for the research prototype solution.

Eclipse was a good base IDE because it offered error correction and other IDE specific functionalities. Overall, this provided the basis of a solution which was entirely written in Java and could be packaged up if Maven was used to resolve dependencies.

The functionality available allowed the merger of results from different webpages into a dictionary structure which could be used to store an animal and its related properties as well as an animal class and its associated properties. Given the generated dictionary, the program could then convert the dictionary keys and values into an ontology of classes (keys) and their properties (values). This allowed the reasoner component to produce the inferred information using the input from the dictionary structure.

The implemented classes were tested individually and integrated to form the final solution. The correct implementation of the iteration requirements was validated and that the code was well written. The code also featured a small amount of configurable code to extend its use for any domain. The final important activity at the end of the iteration was to reflect on what had gone well and what had not. This allowed one to see what could be improved and how best to improve the end result for similar projects of this form.

One particular example of this was the work done towards the end of the project on implementing “food chains”. The development work in the previous iterations on “information extraction” and “ontology creation” contributed to a much deeper understanding of the overall problem domain. This made it much easier to implement additional functionality using previously designed and documented code modules.

Section 4 – Problem Specification & Design

A project deliverable, which was compiled incrementally over a number of iterations, was the requirements and acceptance criteria specification which is documented in Appendix A. This was an essential step to visualising the problems and realising what could be done to achieve the required functionality.

The implementation of the core requirements and the fact that the early iteration passed its acceptance criteria meant that further investigations of other interesting concepts could take place. Many of these had been envisioned when completing the early development iterations. A number of *Supplementary Requirements* arose from this later research to explore the problem domain further.

Section 4.1 - Problem Specification

The requirements for this prototype solution are detailed below, as well as the assumptions made. Additionally, risks for the project were considered, documented and mitigated.

Section 4.1.1 - Assumptions

The following assumptions were made for the prototype solution:

- Development effort would be focused on implementing core functionality rather than on beautification of the user interface.
- Top level Animal classes are one of “Bird, Reptile, Amphibian, Fish and Mammal”
 - It was appreciated that an animal such as a fish could also be sub-classified as “*Actinopterygii*” or “*Chondrichthyes*”. Such subclasses will be ignored.
 - More elaborate edge cases are out of scope e.g. amoeba, protozoa etc.
 - Dinosaurs will also be ignored but will become an area for investigation.
 - Other classes which are being ignored are Insects and other more complex animal cases such as “platypus”.
 - The initial development iterations will consider animals which are recognised to be of that class e.g. Lion:Mammal, Ostrich:Bird.
- All animals which have a certain set of defining properties will be classified as belonging to a certain animal class.
 - e.g. all animals with wings are inferred to be Birds.
 - A subset of these animals may require further classification properties for them to be correctly classified as that animal. As this is a concept demonstrator, this was not deemed to be critical.
- Web-pages used for extraction contain correct and relevant animal data although it is appreciated that some webpages may contain missing data.

- Animal properties will initially be extracted from a look-up table with the opportunity for the system to extract these itself in later iterations.
 - More complex animal properties such as “vertebrates” vs “Invertebrates” could be examined in future iterations.
- The websites used during the prototyping process will still be available for future enhancements.

Section 4.1.2 - Supplementary Requirements

These “Supplementary Requirements” were added to cover additional optional features identified during problem specification.

Req. ID	Requirement Description	Risk	Acceptance Criteria	Acceptance Method	Notes
S-20	The system should generate different lists of animals.	Low	Verify that a list of animals can be correctly generated.	Inspection	Basically allows modifiable input to the program.
S-21	The system should get the HTML for a given page.	Low	Verify that given an input URL, the system will correctly return the HTML for a given page in “Document” format.	Inspection	An essential feature identified early on.
S-22	The system could check the animal classification against expected results.	High	Verify that an animal has been classified correctly and if not standardise the result.	Inspection	This is to combat user error and check that the animal classification is correct.
S-23	The system could also look for the scientific names of Animals which can be added to the Ontology.	Low	Verify that scientific names can be correctly added to an Ontological model.	Inspection	Proves that properties can be easily added and extensibility to other domains is possible. e.g. “Lion” hasScientificName “Panthera Leo”
S-24	The system could ask for user input for unclassified results.	Medium	If the class cannot be found verify there is a facility to add manually.	Inspection	Essential feature identified

Section 4.1.3 - Risks & Mitigation Strategies

Eliminating or reducing risk is an important part of the iterative approach. The following table details the risk, mitigation techniques and risk likelihood. Taking risk into account made it easier to prioritise based on the most important requirements.

The greatest risks were from the “Short Timescale” and “New Implementation Technologies” which were being used for the first time. There was a significant risk that these technologies would be difficult to adopt. The timescale was relatively short considering the development effort and documentation tasks.

Risk Description	Mitigation Strategy	Risk Level
Development Complexity - complex & innovative project.	Agile development allows for complexity to be catered for during each iteration. This ensures the system is built up over time and allows for more innovative ideas to be implemented.	Medium
Short Timescale – the project was time limited.	Agile development allows for more flexibility as features are incrementally implemented in each iteration. Smart planning mitigates this and ensures time is spent efficiently.	High
New Implementation Technologies - development technologies such as Jsoup had never been used.	Detailed study should provide the necessary understanding to use them. Through smart prototyping and agile development this risk is mitigated.	High
Similarity Comparison Function - Required a function to compare string similarity.	Implemented solutions were assessed and a well-documented solution was found.	Medium
Specific Functionalities Required – functionality for a lot of complex system behaviours.	Mitigated through looking at currently implemented solutions and looking for modules which achieve the required functionality. Often the methods were not available but used as a study resource.	High
Program Quality – the system is required to be robust and maintainable.	Through frequent and iterative testing techniques such as “regression testing”.	Medium
New Development Methodology – the iterative methodology was unfamiliar	Research into what makes an iterative approach successful. The author had experience of similar development tasks in the past albeit in a less formal iterative manner.	Medium

Section 4.2 - Design

The deliverable for the design stage was the class diagram documented in Appendix B. It shows the class relationships and the methods they implement. Although the system architecture has evolved, the same basic methods are implemented with additions benefitting program quality.

A sequence diagram was also developed to show interactions (inputs/outputs) between classes and is provided in Appendix B.

There was no need to design a complex GUI interface for the prototype as the focus was on finding out whether the proposed ideas were feasible when applied to the domain of “*animal classification*”.

Section 4.2.1 - Design Decisions and Rationale

The following tables detail the more important design decisions made during the 4 development iterations. They also provide the rationale and key ideas behind why a given approach was adopted.

To provide visibility of the degree of completeness in implementing a certain requirement the following codes appear in parentheses following the requirement ID number. This approach allows requirements to be implemented incrementally as precise details are discovered.

Completion Code	Definition
Initial	When a requirement is partly implemented in an iteration
Intermediate	In early iterations, when the implementation of a requirement is being improved upon
Final	When a requirement has been fully implemented

The most important decision decisions are outlined below (see Appendix B for more design decisions and associated rationale)

Iteration 1		
Requirement ID & Brief Summary	Design Decision	Rationale
F-1 (Initial), S-21 (Initial) - Extract webpage HTML data for a given animal.	Extract HTML data using the Jsoup method <code>:Jsoup.connect(URL).get();</code>	Jsoup is a well-documented and suitable library for “information extraction”.
F-2 (Initial)- Extract the animal’s classification.	Extracted by looking for the text of “Class:” using Jsoup “ <code>.select()</code> ”. Then looking within the HTML to find the element which conforms to this criteria to find the text associated with this element.	An initial test to learn how easily “information extraction” could be achieved. Later approaches focused on extracting properties to infer an animal’s classification. Extracted classification information is still useful when properties cannot be found.

Iteration 2		
Requirement ID & Brief Summary	Design Decision	Rationale
F-2 (Initial) - Extract useful properties from the HTML data.	Use of Jsoup's ".select()" method to find HTML which matches word patterns on a basic initial level.	Provided all the "information extraction" capabilities required e.g. looking for strings which contained certain words such as "wings" and then adding these properties.
D-10 - (Final) Model extracted properties as an ontology.	To use a set of developed methods to create the ontology and relationships.	Produced more modular code and concealed solution complexity. It was implemented in different functions rather than in a large block of code (as it had been in the previous iteration).
F-13 (Final) – Create key value pairs for a given animal and their classification.	More complex key value pairs were constructed for both an animal's classification and its properties.	A simple way to use the "split" function to separate out different pieces of information. Through splitting by comma for one of the values of an entry, properties could be obtained.

Iteration 3		
Requirement ID & Brief Summary	Design Decision	Rationale
F-2 (Final)- Extract properties in a more sophisticated manner	To use a more specific set of classification words to search and define a property as being present.	A simple way of demonstrating the concept. There is a risk because there are edge cases with properties, such as "feathers", which are not classified as Birds. This requirement, was more to prove that a refined choice of words could work.
F-3 (Final) - Model these properties in an ontology	To create functionality to add "equivalent" or "restricted" ontology relationships for either an animal classification or animal respectively.	The same code was not being repeated and delivered a more effective function.
F-9 (Final)- Animal Counter object voting system.	Functionality to find the most likely classification from results which include identified classifications as well as unclassified ones.	Important to consider cases where an animal cannot be classified to make the system more robust. N.B. The system will always default to classified values when the voting process completes.
S-22 (Initial)- Checking the animal has an accepted classification.	Implement functionality to check whether the result is similar to an accepted set of classifications.	The Levenshtein distance measure to compare Strings was used.

Iteration 4		
Requirement ID & Brief Summary	Design Decision	Rationale
F-2 (Final) - Extract properties in a more sophisticated way.	To build upon the previous work for this requirement to allow for capitalisation to be taken into consideration.	A more robust system as a result. When comparisons were made, the lower-case function of Java was used. A scenario of no properties being extracted was factored in.
F-16 (Initial) - Create food chain of animals and the animals they “eat” and are “eatenBy.”	To extend the recursive process and make “eats” the inverse property of “eatenBy”.	The system could infer that if a given animal (Lion) is the predator of some animal (Antelope). Then an Antelope is the prey of a Lion.
	The tree creation process considered all animals which “eat” and are “eatenBy” in a more advanced manner.	This even more recursive approach meant that additional information about animals which was stored on the webpage could also lead to further knowledge inference.
S-23 (Final) - Allow the scientific names of animals to be added.	If the “ScientificName:” field does exist on a given page, then extract this using Jsoup’s “.select()” method.	The previous extraction approach attempted was a test to see how easily new properties could be added. It was easy to extract properties if the pattern was known.
S-24 (Final) - Provide human help when a given animal cannot be classified.	If the system cannot retrieve a valid animal classification then the system will ask for user input.	Involves the user but their actual processing requirements are small. The case statement only retrieves input if a user wants to provide classification information.

Section 4.2.2 - Initial algorithms designed

Pattern Extraction (Early prototype version)

```
Def patternExtractor (Document extractedHTML):
```

```
    pattern = "table[class = infobox biota] tr:contains(class:) a[title]"
```

```
    interimClass = extractedHTML.select(pattern)
```

```
    item = interimClass.attr("title");
```

```
return item
```

Initially designed to extract and return information from a HTML Document format by using a pattern to search for “interimClass” text inside a given page. The “interimClass” is examined and the “title” attribute determines the animal’s classification.

Merge Dictionary of Results (Early prototype version)

```
Def mergeResults(Dictionary animalDict):  
    valueToBeAdded = "NONE"  
    For (entry : animalDict):  
        values = entry.getValues().split(",")  
  
        For (value: values):  
            valueToAdd = value  
  
            if (valuesDict.containsKey(valueToAdd):  
                valuesDict.put(valueToAdd, valuesDict.get(valueToAdd) + 1)  
  
            else:  
                valuesDict.put(valueToAdd, 1)  
  
        largestCount = 0  
        largestKeyCount = ""  
  
        for(numberResult : valuesDict):  
  
            if(numberResult.getValue() >= largestCount):  
  
                if(!numberResult.getKey().equals("NONE")):  
                    largestKeyCount = numberResult.getKey()  
                    largestCount = numberResult.getValue()  
  
        animalDict.put(entry.getKey(), largestKeyCount)
```

The key purpose of "mergeResults()" is to take a set of proposed "*animal classifications*" and count the number of times each classification occurs. Based on the "valuesDict" the most likely classification can be determined with results such as "NONE" being ignored for the "voting process".

Section 5 – System Implementation & Class Descriptions

Section 5.1 - System Implementation

The following section provides details of the core techniques which were used to construct the main functions of the prototype system. Exemplar screenshots of the technologies and implementation approach are provided in Appendix E.

The key modules include: “InformationExtraction.java”, “OntologyCreation.java” and “Food Chain.java” and are detailed below together with their associated classes for clarity.

Section 5.1.1 - Key Functionality Details

Key Functionality	Associated Java Classes	Class Description
<p>Information Extraction - extracting the properties for each animal found in the list of test animals.</p> <p>The animal list is used as an input by these Java classes to perform the process of “information extraction”.</p>	InformationExtraction.java	<p>Functions to extract data from HTML content based on the input of a list of animals</p> <p>The list of test animals is also created by this class.</p>
	AnimalClassExtractor.java	<p>Extracts a given animal’s classification and properties and a dictionary is formed from the results.</p> <p>Results are extracted based on the pattern which is present in one of the candidate animal URL pages.</p> <p>The values for a given animal are then completed by adding the animal’s properties which are extracted from the PropertyExtractor.java class.</p>
	AnimalCounter.java	<p>“Data structure” object and class to take a number of retrieved results from web sources and count the results to determine the most likely classification.</p> <p>This is performed by the “mergeResults” function of this class. This class also offers methods to add to the dictionary structure.</p>
	AnimalClassSimilarity.java	<p>This class takes the result of a given animal classification and standardises it based on whether the extracted word is similar to one of the accepted words.</p> <p>Achieved through the “Levenshtein distance” measure.</p>
	PropertyExtractor.java	<p>Builds up a String of the properties which a given animal possesses, which are then appended to the dictionary structure in AnimalCounter at the “mergeResults” phase.</p> <p>If the extractedHTML does not contain the animal’s name then “NOPROPERTIES” is returned instead.</p>

<p>Ontology Creation</p> <p>-Creating an ontology of animals and their animal classification and associated properties.</p> <p>This class is designed to allow the inference of previously unknown information.</p>	<p>OntologyCreator.java</p>	<p>Convert properties which have been extracted at the “information extraction” stage into an ontological format. OntClasses are created for animals and animal classifications.</p> <p>Properties are added as ObjectProperties and each of these classes and property relationships are then added to an OntModel object which represents the created model. All of these classes are JENA specific.</p> <p>Methods provide model inference using “Pellet” as a reasoning tool. The animal’s classification properties or the animal’s properties are added to the properties. This ensures that an animal with “wings” can be inferred to be a “bird”.</p>
<p>Food Chain Hierarchy Creation</p> <p>- create a hierarchy of which animal “eats” and is “eatenBy” another animal.</p>	<p>FoodChain.java</p> <p>This class was implemented after “InformationExtraction” and “OntologyCreation” classes were tested and working as expected.</p>	<p>Combines both the classes and functionalities of “information extraction” and “ontology creation” as well as recursive techniques to produce a recursive routine and a set of hierarchical relationships with scope for future enhancements.</p> <p>Takes as input a “startAnimal” and traverses extracted results using the “InformationExtraction.java” class to retrieve animals which are a predator and a prey of that given animal. Recurses through all the animals retrieved.</p> <p>Animals with their associated “eats” and “eatenBy” relationships can be built into the ontological model using the “OntologyCreator.java” class.</p>
<p>Testing- methods for testing the solution.</p>	<p>Testing.java</p>	<p>Provides functions for testing based on results which are sent to be output to the console in the form of standard output.</p>

Section 5.1.2 - Information Extraction Approach

The “information extraction” capabilities took advantage of the basic input/output capabilities of “Jsoup”.

The initial step was to create a list of animals to be tested for and this list was then passed to the *animalClassExtraction* function. The purpose of this being to extract the class for a given animal based on the classification data stored on a specific website. This data was returned as an *AnimalCounter* object which implemented the “merges” method.

The system took this *AnimalCounter* object e.g. {[Lion:Mammal,Mammal]} (which is a complex dictionary) and “merged” the information to get the animal and its associated “expected classification” e.g. {[Lion,Mammal]}.

The properties of a given animal are then appended to the values for a given “animalKey”: made up of an animal name, a delimiter (“#”) to separate content and the “expected classification”. Property extraction is based on whether a certain set of words are specified on that animal’s web-page. This is a limited approach but once again the focus was based on the inference capabilities of JENA over “information extraction”.

```
Animal:Lion
https://en.wikipedia.org/wiki/Lion classified: Mammal
http://a-z-animals.com/animals/Lion classified: Mammalia
http://animals.sandiegozoo.org/animals/Lion classified: Mammalia (Mammals)
Lion

Mammal
Animal:Lion
Lion-Class to be added:Mammal
Scientific name cannot be found:http://animals.sandiegozoo.org/animals/Lion
Scientific name added:Panthera leo from source:http://a-z-animals.com/animals/Lion/
{hairy#http://animals.sandiegozoo.org/animals/=true, vertebrate#http://a-z-animals.c

Animal Dictionary contains
{Lion#Mammal=hairy,hasScientificName$Panthera leo}
Final animal dictionary of properties: {Mammal#ANIMALCLASS=hairy, Fish#ANIMALCLASS=g
```

Figure 3: Screenshot For A Lion (Information Extraction Process)

Additional class entries were later added so that a given class depends on a certain property “[Ostrich#Bird: wings], [Bird: wings]”. This means that the *OntologyCreation.java* class can make use of this important classification information.

Section 5.1.3 – Ontology Creation Approach

The capabilities for “ontology creation” came from JENA. The ontology creation class took results which were obtained from the “information extraction” phase such as {[Lion#Mammal,hairy]} and then added the properties (such as “hairy”) to a specified model.

```
Properties for:Lion:[hairy, hasScientificName$Panthera leo]
Should be classified as:Mammal
Animal being added:Lion
Property being added:hairy
Property being added:hasScientificName$Panthera leo
Split result[hasScientificName, Panthera leo]
```

Figure 4: Properties Added For A Lion (Ontology Creator Process)

The animal’s “expected classification” information (e.g. “Bird”) was ignored at this stage because it is what the system was trying to infer. It was kept because in future iterations, it could be useful to check that “*animal classifications*” were being correctly inferred or added to the classification when properties could not be found.

This “ontology creation” process begins by creating an ontology from a baseURI and defining the top level “ANIMAL” class. When a list of properties are passed across, the system adds these to the ontology. It accesses these properties by “splitting” by “,”, which works because each animal in the properties list is comma separated. For each property, the system looks at whether the key for that entry is an “animal class” or an animal itself.

Depending on this, the system either adds an “equivalent class relationship” (animal class) or a “restriction on some property” (animal). It does this by creating the relevant *ObjectProperties* and also the *OntClasses* themselves.

```
Properties for:Mammal:[hairy]
Should be classified as:ANIMALCLASS
Class being added:Mammal
Property being added:hairy
```

Figure 5:“ANIMALCLASS” Properties Added For A Mammal (Ontology Creator Process)

Figure 7 represents the created model for a Lion and details the important classes used to infer that the “animal classification” of a Lion is a “Mammal”.

```
<http://www.example.com/seanjames/ontologies/animals#Lion>
  a    owl:Class ;
  rdfs:subClassOf <http://www.example.com/seanjames/ontologies/animals#Animal> ;
  rdfs:subClassOf
    [ a    owl:Restriction ;
      owl:onProperty <http://www.example.com/seanjames/ontologies/animals/hairy> ;
      owl:someValuesFrom <http://www.example.com/seanjames/ontologies/animals#Animal> ] ;

<http://www.example.com/seanjames/ontologies/animals/#hasScientificName>
  <http://www.example.com/seanjames/ontologies/animals#Panthera leo>
, <http://www.example.com/seanjames/ontologies/animals#hasScientificName> .

<http://www.example.com/seanjames/ontologies/animals#Mammal>
  a    owl:Class ;
  rdfs:subClassOf <http://www.example.com/seanjames/ontologies/animals#Animal> ;
  owl:equivalentClass
    [ a    owl:Restriction ;
      owl:onProperty <http://www.example.com/seanjames/ontologies/animals/hairy> ;
      owl:someValuesFrom <http://www.example.com/seanjames/ontologies/animals#Animal> ] .
```

.....

Inferred model:

```
<http://www.example.com/seanjames/ontologies/animals#Lion>
  a    <http://www.example.com/seanjames/ontologies/animals#Animal> ,
<http://www.w3.org/2002/07/owl#Class> ;
  <http://www.w3.org/2000/01/rdf-schema#subClassOf>
    <http://www.example.com/seanjames/ontologies/animals#Mammal> ;
  <http://www.example.com/seanjames/ontologies/animals/#hasScientificName>
<http://www.example.com/seanjames/ontologies/animals#Panthera leo>
, <http://www.example.com/seanjames/ontologies/animals#hasScientificName> ;
```

Figure 6: Inferring A Lion To Be A Mammal(Ontology Creator Process)

Section 5.1.4 - Recursive Approach: Creating a Hierarchical Food Chain of Animals

Successfully completing the core functionality of the system in the initial development iterations meant that work could begin implementing other desirable requirements such as “Requirement 16”: *“Create a hierarchy based on the ‘one animal’ ‘eats’ ‘another animal’ relationship to create a food chain of animals.”*

This is a novel concept, unique to the domain of “animals” and their associated properties. The idea is to programmatically build a food chain, where a given animal eats another animal. This is powerful because if the system can retrieve a few animals which a given “startAnimal” “eats”, then the system can get the animals they eat and the animals they eat and so forth. Recursing through each of these possibilities builds up a large food-chain hierarchy.

- The “inverseOf” property was used when performing inference because if a given animal “eats” another animal, then the other animal in turn is “eatenBy” that animal.
- Additionally, properties such as “transitive” relationships could be applied given that an animal was known to be at the top of the food chain. This would mean that other properties of “eats” could be inferred if an animal “eats” just a few animals. It is likely if a Fox can eat a Mouse and a Lion “eats” a Fox, then a Lion can also “eat” a Mouse.

Figure 3 shows an example portion of the “Food Chain” hierarchy for a Lion:

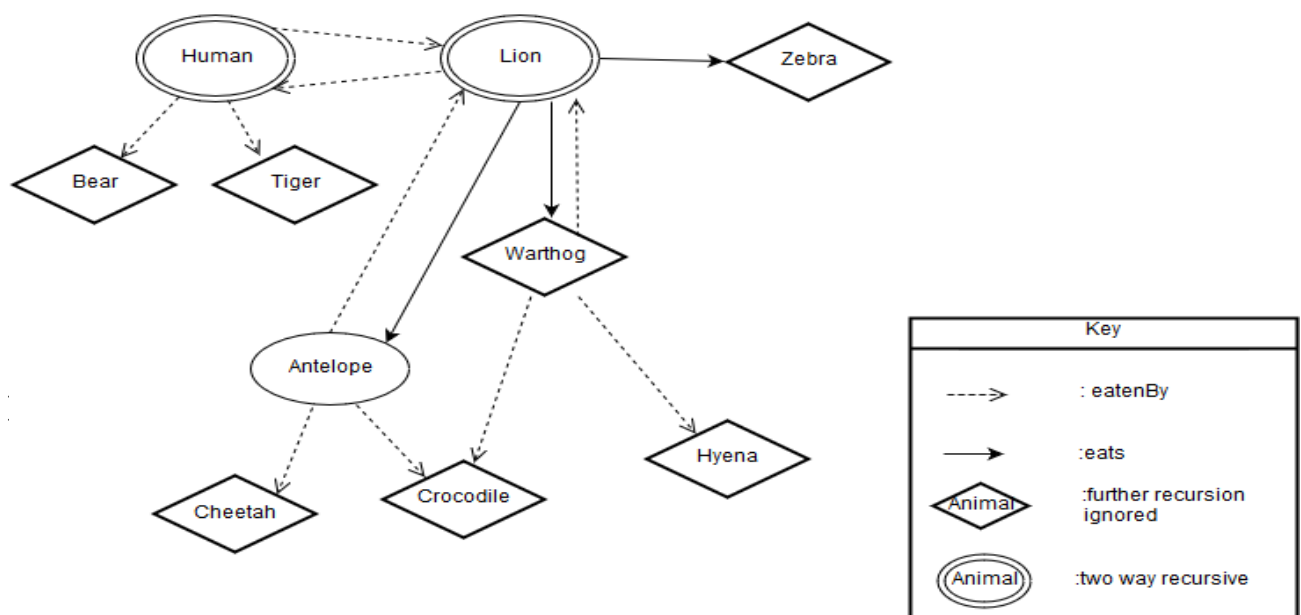


Figure 7: Lion Food Chain Hierarchy Representation

Section 5.1.4 - Recursive Algorithm Design

Accordingly, a recursive algorithm (detailed below) was designed and implemented, to retrieve the animals which are a predator and prey of another given animal. It recursively applies this principle for each of these possibilities where a given option has not been explored before.

Java Algorithm: Adding an Underscore To Animal Names With Spaces

The algorithm checks whether the target animal has any spaces between its name and if it does, the program adds an underscore so that when the recursion occurs for that animal the URL which is passed is valid.

```
if (Arrays.asList(animal.split(" ")).contains(animal) == false){  
    animal = String.join("_", animal.split(" "));  
}
```

The program retrieves the HTML for the chosen animal. Then the system extracts the predators and prey of that animal, looking in the HTML code where a table row contains the word “prey:” or “predator:” (errors are caught if no information is found).

```
preyHTML = InformationExtractor.patternExtractor(extractedHTML, "table[class =  
article_facts] tr:contains(pre:) td");
```

Recursive Algorithm Description

A seeding animal (e.g. Lion) is first passed as an argument to the “animalRecurse()” function to fulfil the criteria for both the “eats” and “eatenBy” recursive search processes.

The simplified pseudocode for the core algorithm is as follows:

Function animalRecurse(animal):

While NOT all-predators-and-prey-found-for (animal):

extractPredatorsAndPrey(animal)

IF not all-prey-processed (animal):

identify-next-animal-prey (animal, prey)

add-prey-information-to-model (animal, prey)

animalRecurse(pre)

IF not all-predators-processed (animal):

identify-next-animal-predator (animal, predator)

add-predator-information-to-model (animal, predator)

animalRecurse (predator)

In the case of animals identified during the “eats” search process (e.g. Antelope, Warthog and Zebra) that the seed animal consumes, each of these is then subjected to its own “eats” and “eatenBy” search processes recursively. In turn, the animals that are discovered through

these search processes are themselves subject to the “eats” and “eaten by” recursive search processes, which continue until there are no more animals to be considered.

A similar search process happens for animals which the animal is “eatenBy” e.g. a warthog is “eatenBy” a cheetah. The cheetah is then passed as an argument to the recursive search function which identifies both what it “eats” and in turn what it is “eatenBy”, which are themselves parameters passed into the recursive search function in the same way, until there are no more animals to be considered.

Once these checks have been performed, it is important to check whether the predator/prey which has been extracted is an acceptable “animal”. This check is required because some animals eat fruit or vegetables. These are considerations which have not been added to the solution because they are not animals. End conditions are added to the recursive routine to prevent it from looping infinitely.

Finally, the properties can be added through a function which interacts with the “OntologyCreator.java” class to create the properties “eats” and “eatenBy” respectively. It also creates animal classes and its predator or prey relationship. This process is then repeated for the rest of the search processes.

Section 5.1.5 - Important Algorithm Descriptions

This section details the most important algorithms which are used by this program to perform “information extraction” and the “ontology building” processes. Some algorithms have been included alongside the more advanced prototyped methods. Appendix C details more detailed algorithms for these key modules.

The more sophisticated algorithms have evolved over time due to further insight into the problem area, as well as understanding how similar problems could be solved more effectively in the future. This was done either to facilitate efficient testing of many different outcomes or to build up better “information extraction” and “ontology building” methods.

Creating a basic animal list test set

This was a simple algorithm to populate an input list of animals for the program to get classification information and properties.

Function animalListCreator():

```
    animalList = “Lion,Ostrich,Frog,Eagle,Crocodile” (Add animals)
```

```
    animalList.add(“Monkey”) (add more test animals)
```

```
return animalList
```

Modified algorithm (to create a more sophisticated test set)

The algorithm was modified to get a different spread of animals; this was a useful evolution for future versions of this prototype solution. Use was made of an A-Z list of animals and the Jsoup “.select()” function to select the specific table which contained the required animal. A test animal was added to the list using a for-loop which added a sample of animals.

Function animalListCreator():

```
i = 20
animalDocList = getHTML(http://a-z-animals.com/animals/)

elements = animalDocList.select("table[class=article_az] ul li a")

for(element : elements):

    if(i % 250 == 0):

        animalList.add(element.attr("title").toLowerCase().replaceAll("\\s", ""))

        i = i + 1
```

return animalList

This approach selected the animals from an A-Z list on a website stored in the table class of "article_az", although this same ".select()" method could be adapted for any given page.

These animals were then sampled through the use of a loop which selected any animals which had an "i" value which was a multiple of 250 using a "mod" operation. The counter increments after each animal and this is how they are iterated through. This method could in theory be set to produce an animalsList for a specific test set of many different forms. Using "i % 2 == 0" as the "if condition" would allow even-numbered animals for instance.

The advantage of this latter approach is that the names of animals are taken directly from the website. If the "information extraction" process is occurring partly on that website and their animal pages, you are more likely to get some results returned. i.e. the website should have an animal page for a given "A-Z animal". There is no guarantee that other sites will however.

Similarity Comparison Tool: The Levenshtein Distance

When investigating how to compare the similarity of two strings, the Levenshtein Distance algorithm was selected. More specifically, the "StringUtils" Java library and associated "StringUtils.getLevenshteinDistance()" function. This returns an integer between 1 and 0 based on String similarity.

Given information about dissimilarity, one could work out similarity and turn this into a percentage value. This is what the "checkClassification" method accomplishes and this can be extended to perform any String comparison.

Checking how similar two Strings are**Function checkClassification(String1,String2):**

```
int overallLength = AVERAGE(animal1.length + animal2.length);

int distance = StringUtils.getLevenshteinDistance(String1,String2);

double percentage = (double) distance / overallLength * 100;

return 100 – percentage;
```

This feature was useful for clarifying whether one animal class was close enough to another accepted animal class ("Reptile", "Mammal" etc.) to be classified as that class of animal. This could be extended to compare if the class which has been classified is an accepted class.

Section 5.1.6 - Implementation Challenges

- The system would find it difficult to handle a situation where several similar acceptable classifications were appropriate.
 - A classification such as "Bord"; then "Bord" and "Bird" are similar enough to cause problems.
- The "FoodChain.java" class currently runs based on the "a-animals site" in which a finite number of animals are represented. If more animals are represented, then close attention would need to be paid to stop the recursion from running excessively long.
- The way that languages are formed makes extracting information from text difficult.
 - E.g. Some mammals have fins such as Whales
 - An incorrect system would decide that all mammals have fins.
 - It is far easier to use clarifying words to determine a word's context.
 - "class:Mammal", "property:fins" for a Whale is clearer and easier to understand.
- The results are dependant on the validity of the website sources chosen for "information extraction".
 - When a class or property cannot be extracted, the system becomes less helpful. However, through taking several results the risk is reduced.
 - In the future: where the class cannot be obtained, properties can lead to the inference of the class and vice versa.
- Websites may use different URL syntaxes.
- The availability of a wild-card function would be useful but time-consuming to implement. e.g. "Snapping*turtle" rather than "Snapping turtle", "Snapping_turtle", "Snapping-turtle".
- Erroneous edge cases are difficult to classify due to conflicting properties e.g. "Flying-fox" which is a flying mammal with wings and not a bird.

Section 5.2 - Detailed Class Descriptions

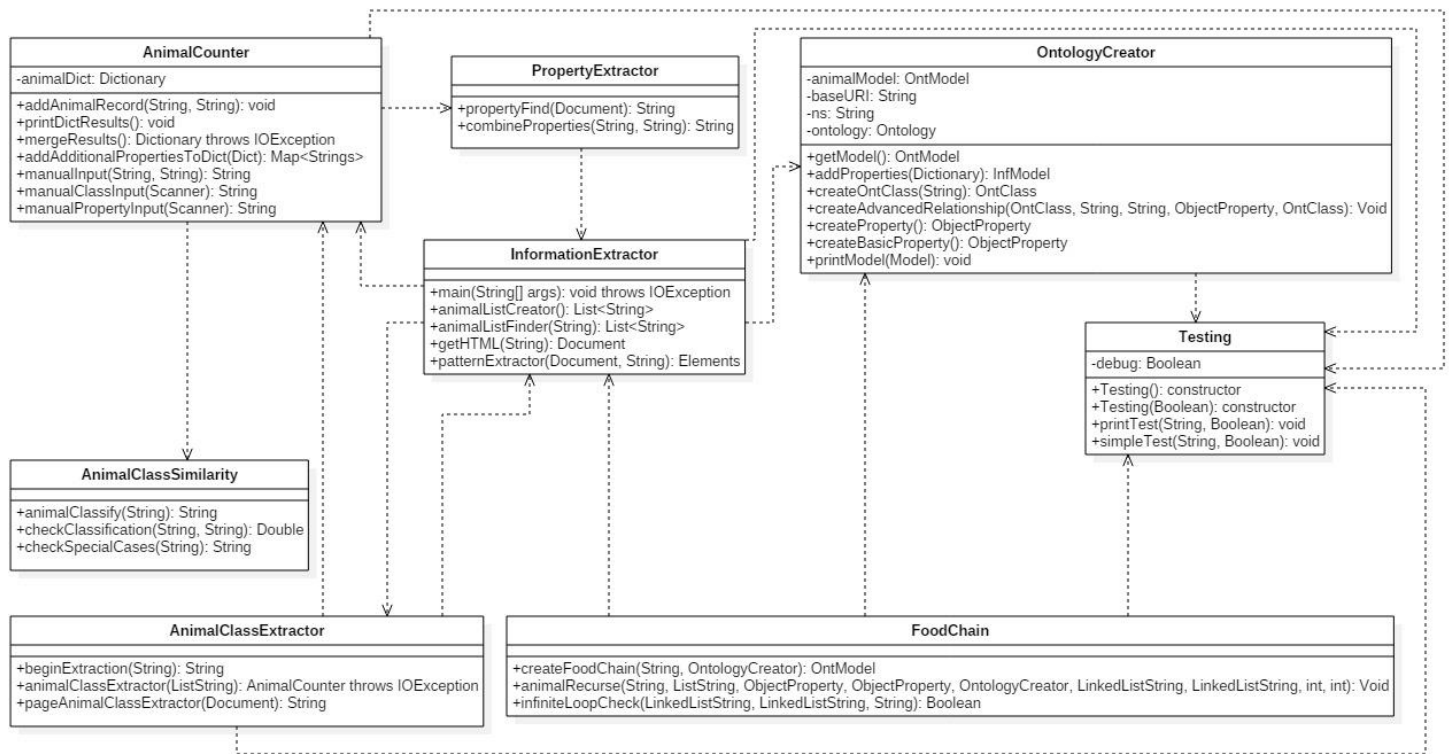


Figure 8: Appendix B - UML Class Diagram (Provided To Show Class Relationships)

Section 5.2.1 - InformationExtractor.java

Purpose was to run the other functions which retrieve, extract and structure the webpage HTML information. It also creates test lists of “animals” as inputs for the system. This is a useful class for testing purposes because certain system features can be disabled.

InformationExtraction.java consists of the following key public methods:

- **main(String[] args): void throws IOException** –this is the first method to run during program execution and is where all other functionality is invoked from.
 - Handles enabling the org.apache.Log4j logger, which is required for Jsoup to provide logging messages. The key method of “animalListCreator” can be used to set up different “animal lists” to test the program further.
- **animalListCreator(): returns List<String>** – creates a list of test animals, specifically as Strings, representing real animals to test the system. This function returns the result as a “List<String>” object. Further animals can be added through “animals.add()” which is a list-specific method chosen to reduce any array position problems.
- **animalListFinder(String URL) : returns List<String>** – extracts all the animals which are on the “A-Z” page of the webpage “a-zanimals.com”

- By modifying the pattern other “A-Z” animal lists can be built.
- **getHTML(String URL): returns Document** – Extracts the entire HTML content for a specific URL and returns as a Document.
 - Traps if the URL cannot be accessed by stating that there is a “Jsoup URL ERROR: Page could not be accessed:” and then prints the URL for debugging purposes.
- **patternExtractor(Document extractedHTML,String pattern): returns Elements** – based on the input of a piece of HTML and a String pattern which is being looked for, this function returns the Elements which conform to these parameters.
 - Follows the syntax of “return extractedHTML.select(pattern)”

Section 5.2.2 - PropertyExtractor.java

Designed specifically for extracting the animal properties present in several webpages. The relevant properties are retrieved and built up into a String to be returned to the program. This class also adds the animal’s “scientificName” if one can be retrieved.

PropertyExtractor.java consists of the following method:

- **propertyFind(String animal): returns String** - achieves property extraction for a given String animal by looking for a specified set of words, which uniquely define a different sort of animal classification.
 - If no properties are found then “NOPROPERTIESFOUND” is returned.

Section 5.2.3 - AnimalClassExtractor.java

Class which contains the functionality to get the HTML code for a given URL specifically relating to an animal’s classification.

AnimalClassExtractor.java consists of the following key public methods:

- **beginExtraction(String URL): returns String**- begins extraction and returns the animal’s extracted class.
- **pageAnimalClassExtractor(Document extractedHTML): returns String** - extracts the classification of an animal based on extractedHTML specific to that animal.
- **animalClassExtractor(List<String> animals) : returns AnimalCounter throws IOException**- given a list of animals, it builds up string classifications for that animal and returns these as an “AnimalCounter” object ready to be “merged()”.

- E.g {Lion, [Mammal, Mammal, Reptile]}, can be processed to infer that a Lion is most likely to be a Mammal.

Section 5.2.4 - AnimalCounter.java

Maintains a counter of the number of animals which are stored in the TreeMap structure. It is a data structure object to handle the way the animal dictionary functions.

AnimalCounter.java consists of the following key methods and attributes:

- **Globally declared attributes**
 - animalDict : Dictionary - Dictionary used to store an animal and the likely classifications it may have. e.g {Lion,[Mammal,Mammal,Reptile]}
- **addAnimalRecord(String animal,String property) : void** - adds an animal and a property of some form to the animalDict object declared globally.
 - If the key is already stored then the updated value will be equal to the previous value concatenated with a comma and the new specified property.
 - Otherwise, the animal and property are placed into the dictionary as normal.
- **printDictResults() : void** – prints each entry and associated key value pair stored.
 - Mainly used for testing purposes.
- **mergeResults() : returns Map<String,String>** – takes an AnimalCounter object and calculates which animal classification is most likely (Bird,Reptile.. etc). The properties for the animal are then added to the “values” for that entry.
 - Important because sites may offer different classifications; it is through this map-reduce approach that classification can be made more accurate.
 - This reduction process is conducted for all the entries stored in the dictionary.
 - The dictionary entry will be updated with the associated classification and then combined with the important extracted properties.
 - Given an input of {Lion,[Mammal,Mammal,Reptile]}, the function will calculate that Mammal came up twice and reptile once so {Lion,Mammal} will be added to the dictionary once all animals are classified successfully.
 - This function will throw an IOException if the animal cannot be found in the requested URL.
- **addAdditionalPropertiesToDict(Map<String,String>animalDict):returns Map<String,String>** - allows additional classification properties to be added to the

animalDict. This is useful for testing purposes where test data can be added to check that the program is performing correctly.

- [ANIMAL]#ANIMALCLASS signifies that the [ANIMAL] should be an animal class and so the relationship of “equivalent to” should be added to the ontology.
 - e.g {Reptile#ANIMALCLASS,scales} : this means that any animal which has scales is “equivalent to” a Reptile. e.g. a Snake.
- The “{Unclassified#ANIMALCLASS,NOPROPERTIES}” entry was added to represent animals with no extraction property results.

Section 5.2.5 - AnimalClassSimilarity.java

Contains the functionality for checking whether the extracted class can be renamed so it follows English spelling semantics or standardised to resemble a well-known name equivalent.

The following key public methods are supported by this class:

- **animalClassify(String): returns String**- compares the given string to a “valid” set of animal classifications.
 - If the given string matches one of the accepted classifications by greater than 50% then that classification will be returned. This is case insensitive.
 - “Reptilia” will map to “Reptile” because only 2 changes are required out of the average number of total letters.
 - e.g. [Snake, Reptilia] will seamlessly map to [Snake, Reptile] and also [Snake, eptil] will map to [Snake, Reptile].
- **checkClassification(String, String): returns Double**- takes as input two strings and compares them for similarity using the “Levenshtein distance”.
 - The number of changes which are required to change one String to the other is the “Levenshtein distance”.
 - This “distance” is then divided by the total number of characters in both words and multiplied by 100 to get a percentage.
 - Taking away 100 gives a percentage similarity rather than a difference.
- **checkSpecialCases(String): returns String** - used if the “Levenshtein distance” and resulting percentage similarity is less than 50% i.e the Strings are quite different.
 - An entry in the dictionary such as [Ostrich#Aves] will be mapped to [Ostrich#Bird] and Bird will be returned as the animal classification.

Section 5.2.6 - OntologyCreator.java

This class consists of the functionality to build the ontology based on an animal and its associated properties. It features a number of useful functions to achieve ontology related actions such as adding properties and creating OntClass objects.

OntologyCreator.java consists of the following key private attributes and public methods:

- **Globally privately declared**
 - animalModel : Ontmodel
 - Ontology model based on a Pellet reasoner.
 - baseURI : String
 - ns : String
 - ontology: Ontology
- **getModel() : returns OntModel**
 - Get globally declared animalModel.
- **addProperties(Map<String,String> animalRelations): returns InfModel**
 - Add obtained properties and animal classification results to ontology
- **createOntClass(String aclass): returns OntClass**
 - Create an OntClass based on a string passed to the function.
- **createAdvancedRelationship(OntClass animal, String type, String v1, ObjectProperty p, OntClass someThing) : void**
 - Create an advanced relationship based on the String type: either as an “equivalent” class or “someValuesFromRestriction” subclass relationship.
- **createProperty(String propertyName,OntClass domain,OntClass range): returns ObjectProperty**
 - Creates an object property with a certain “propertyName” and additionally sets a domain and range to be returned.
- **createBasicProperty(String propertyName): returns ObjectProperty**
 - Creates an object property and returns the result.

- **printModel(Model animalModel): void**
 - Prints the model in its more complex form with all the relationships shown. Afterwards it gets the simplified model as the set of inferred relations.
- **addTriple(OntClass predClass, ObjectProperty property, OntClass PreyClass): void**
 - Adds a triple relation based on a subject predicate and object supplied as parameters.

Section 5.2.7 - FoodChain.java

Prototype function to test the inference abilities of JENA. In particular, can transitive and inverseOf relationships provide the ability to infer further information and create a food chain.

Foodchain.java consists of the following key public methods:

- **animalRecurse(String, List<String>, ObjectProperty, ObjectProperty, OntologyCreator, LinkedList<String>, LinkedList<String>, int, int): void** - Complex function to recursively create a "food chain" of predators and prey and add to the OntologyCreator object and an associated model.
 - Uses a recursive search process to find animals which "eat" or are "eatenBy" a given animal and subsequent animals which "eat" or are "eatenBy" the animal.
- **createFoodChain(String, OntologyCreator): returns OntModel**
 - Initialises variables which the "animalRecurse" method uses for its execution and provides exception handling.
- **infiniteLoopCheck(LinkedList<String> eatsArray, LinkedList<String> eatenByArray, String proposedAnimalRelation) : returns Boolean**
 - Checks whether an animal has already been traversed with respect to either the "eats" or "eatenBy" relationship. If an animal has been traversed then this will return "true" which prevents it from looping infinitely.

Section 5.2.8 - Testing.java

Provides functionality for optionally debugging and printing required tests.

It implements the following private attributes, Constructors and key public methods:

- **Globally declared attribute**
 - Debug : Boolean
- **Testing(): constructor**
 - By default Debug is set to true with this function.
- **Testing(Boolean): constructor**
 - Set debug to the value which is specified as a Boolean.
- **printTest(String, Boolean): void**
 - If the value of debug is set to true then output the String parameter.
- **simpleTest(String, Boolean): void**
 - Simply print out the parameter input if the Boolean value is true.

Section 6 – Testing, Results and Findings

Section 6.1 - Testing

The system was designed to test different principles of “*information extraction*” and it was decided that the system should work for the most important cases. Difficult cases could later be examined and investigated if time was available. Difficult edge cases (such as “*flying_fox*”) were looked at in detail, however they were not formally tested.

The approach for testing, was therefore focused on regular re-testing and checking the output from the console and was performed whenever a code modification was made. From the test results, it can be seen that was a good approach and efficient.

The test plan, unit testing, functional testing, system testing and overall testing approaches are covered in Appendix D and followed an iterative testing paradigm. Unit testing involved taking modules one at a time and performing sets of tests on them. Once this was achieved, modules were integrated and testing was repeated to ensure that code changes did not break the system. Once all unit and integration testing had been completed, system testing was performed.

The results of unit, integration and system testing were generally as anticipated and correct. The system adapted well to difficult and unexpected cases e.g. Dinosaurs or Insects.

Section 6.2 - Results and Findings

All the objectives outlined in Section 1.4 have been satisfied by the developed prototype including the more complex food chain hierarchy goal.

The project has proven that “Information extraction to infer new knowledge” is an entirely plausible concept. “Information extraction” was investigated in the first instance and during research many tools were identified.

The scope was greater than simple information extraction however, and there was the more elaborate need to model extracted information in a machine readable format and to allow reasoning processes to occur.

The “information extraction” tool of Jsoup was used for the concept prototype and provided the core capabilities required. This was the plan all along, providing more time to investigate features which had not been studied before. Other more advanced technologies which could learn from extracted patterns and web-pages visited would have complemented the solution.

Taking the results from several webpages provided a more appropriate classification method. The approach was focused on using dictionaries of key value pairs to describe an animal’s classification and properties. Taking these key value pairs allowed a merge function to be applied, which could take multiple suitable results and apply a “voting system” to determine which result was appropriate.

JENA was chosen for ontology modelling, because it featured substantial functionality to achieve this end. Extracting webpage information to be ontologically modelled was an approach that worked well and the process executed quickly. “Pellet” was a successful way to infer a given animal’s classification based on an animal’s set of determining properties.

The results of ontology modelling were almost always correct for non-edge case animals. This is because the way a rule based reasoner works does not affect the results. If a fox has a property “hairy” and a mammal is an animal which is “hairy”, it is seamless for a reasoner to decide that a Fox should be inferred to be a mammal.

Using a Java class to achieve animal class standardisation was appropriate. Even though a top level known class might be “Fish”, subclasses did exist such as “Actinopterygii and “Chondrichthyes”, which could easily be standardised to “Fish”. This confirmed another idea, namely that a Latin name could easily be standardised to a more “well-known” English name.

Additionally, the “Levenshtein distance” was implemented to check how similar the input was for a specified set of animal classes. This ensured that even if a website featured a spelling mistake the system could adapt. This also showed that with further development, the system could be made robust to such types of mistakes in other related modules e.g. for checking manual input and standardising it amongst other more advanced features.

Later investigations into whether properties could be added, such as a “scientificName” were very fruitful and proved that it would be possible to extract a large amount of information and to add this to a model. Through the use of delimiters, the extracted properties could be “.split()” and added as a triple relationship to a given model.

An interesting outcome of this research and development work is the “food chain” creation module. This module uses one “startanimal” and retrieves all the animals which this animal eats, as well as the animals those animals themselves eat. As a result, it builds up a hierarchy of relationships and then adds these to a specified model.

This is very powerful because in the future, the classification properties and food chain information of each recursively analysed animal could be captured and further modelled. With development, the system could extract animal information and build a food chain from every relevant animal website proving how extensible this initial designed solution is.

Section 6.2.1 – Knowledge Inference Result Exemplars

Animal	Expected Classification	Inferred Classification	Extracted Properties	Some of the program extracted Food Chain information.			Extracted Scientific Name
				Eats(Prey)	Eaten-By(Predators)	Food Chain Size	
Lion	Mammal	Mammal	hairy	Antelope, Warthog, Zebra, Human	Human	111	Panthera Leo
Octopus	Fish	Fish	gills, vetebtrate	Crab, Fish	Dolphin	156	Sphyaena

Zebra	Mammal	Mammal	hairy	<Fruit>	Hyena Lion, Leopard	111	Equus zebra, Equus quagga, Equus grevyi
chinstrap _Penguin	Bird	Bird	wings	Killer Whale, Leopard Seal	Shrimp, Fish	167	Pygoscelis Antarcticus
Crocodile	Reptile	Reptile	scaled	Hyena, Warthog, Wilderbeest, Fish	Human	111	Crocodylus
Albatross	Bird	Bird	wings	Crab, Fish, Squid	Tiger_shark, human	161	Diomedidae
<p align="center">Edge Cases Begin</p> <p align="center">(Assumption: If extracted words are not found in the “A-Z animal” list then these results are invalid.</p>							
Snake	Reptile	Reptile	scaled	-	-	-	Serpentes
Ostrich	Bird	Bird, Mammal	wings hairy	<Fruit>	Hyena, Lion, Cheetah	114	Struthio Camelus
Angelfish	Fish	Reptile	scaled	Fish	Fish, Bird	114	Pomacanthidae
Bat	Mammal	Mammal	hairy	Frog	Snake, Eagle	122	Chiroptera
Frog	Amphibian	None	-	Fly	Fox		Rana Temporaria
bearded _Dragon	Reptile	Reptile	scaled	Insects	Crocodile, Snake, Bird	115	Pogona Vitticeps
T-Rex	Dinosaur	Dinosaur	-	-	-	-	-
Butterfly	Insect	Mammal, Bird	wings hairy	Frog, Bat	-	124	Papilionoidea
Dragon	None	Reptile	scaled	-	-	-	Pogona Vitticeps

Section 6.2.2 – Critical Considerations

Learning. This research and development project has facilitated a tremendous amount of skill-set evolution, in particular, learning and understanding about the problem areas of “information extraction” and “ontology creation”.

Jsoup. Using a basic “information extraction” capability was essential to allow research into more sophisticated ideas and approaches. Not only has this allowed the creation of a better overall result, but has allowed a useful study into how to implement a “food chain hierarchy” in a recursive manner. Jsoup was useful as an “information extraction” tool and is recommended for similar extraction tasks. It was well documented which made adopting its extraction approaches easier

Information Extraction. Jsoup is a great tool for extracting information, the problem is whether this information should be trusted with greater authoritativeness compared to other extracted information. Experts classify animals in a more precise manner down to the

“subclass” level, these results can either be standardised, or they can be added to a more precise ontological model.

Food Chain Creation. The algorithm which performs Food Chain recursion does not add results if they are not part of the “A-Z list” of accepted animals. In general, the largest chain is produced from an animal which is eatenBy a lot more animals at a higher level.

JENA Ontology Creation. JENA was a very useful library resource. However, it was difficult to write code to emulate the “Protégé” Java code that needed to be recreated due to its inferior documentation and exemplars. Much JENA programming involved trial and error and using Eclipse’s built in IDE functionality to find the appropriate method for a given object. The biggest problem was adding “advanced relationships” (such as equivalent class for an animal’s classification). The JENA methods provided an extensible solution appropriate for modelling different extraction domains.

Maven. The use of Maven to handle package dependencies was valuable. This was a key concern for this project and this approach provided a good way to satisfy the dependencies for both Jsoup and JENA.

GitHub. Making use of the industry standard GitHub to maintain code was also valuable as it provided a means to backup, track updates and version control the developed code. An end goal was always to provide a set of well documented modules to take current ideas of information extraction further.

Section 7 – Future Work And Personal Reflections

Section 7.1 - Future Work

The following enhancements provide opportunities for improving the prototype system:

- **Use of commercial search engines.** Choosing suitable webpages automatically from a search engine (such as Google) using their querying facility.
 - For this project, web-pages were selected by inspecting the HTML code behind the page. A smart artificial intelligence agent could perform this task by looking for the required domain classification properties to ensure that relevant extraction results are being returned.
- **Web-site prioritisation.** Developing a method for finding and prioritising suitable websites to scan the most relevant pages.
 - With billions of available websites, the system needs to both find them and filter out inappropriate ones.
- **Context sensitivity.** Extending “Information extraction” algorithms to analyse the specific context of a word given the verbiage it is surrounded by.
 - Currently, if the word is contained in a page and that same word is present and cross references other pages, then the word is deemed to be suitable.
- **Sentence parsing.** Enhancing the food chain creation process by extracting information from sentences that identify predators and prey.
- **DNA fingerprinting.** Using genome data to map each animal with the specific DNA that animal has and then look for further patterns.
- **Graphical analysis.** Extending current capabilities to parse useful graphical information as well as text from HTML.
- **Other languages.** Extend to include foreign language websites.
- **Manual assistance.** Allow the system to get help from a human assistant.
- **Mind-mapping techniques.** System to produce mind-maps of captured information.
- **Food Chain Improvements.** Add further sophistication to extract whether results should be regarded as authoritative or not.
 - Cross reference extracted results
 - Uncertainty scoring system could be added to evaluate how reliable a site is.

Section 7.2 – Personal Reflections

When considering a dissertation proposal topic, I wanted to take on a project which would be challenging but achievable in the timeframe. I was also looking for a useful research theme to provide additional insight into interesting and current problem areas.

When I came across Dr. Andrew Jones' "information extraction" proposal I was immediately interested and wanted to find out more. I realised that there was some positive benefit to be gained from researching into this topic.

Making use of what was learned in lectures for "Large Scale Databases", meant key techniques were already appreciated. Specifically, ontological principles such as classes and adding properties.

After investigating what had already been achieved, it was decided that the proposal would need to be modified slightly. This made it more aligned with what I wanted to achieve and beneficial to investigate. The result was that the title and emphasis of the project changed to "*Information extraction to infer new knowledge*". Once these modifications were made to the proposal, this motivated me further and kept me focused and interested because of the novel and useful nature of the work.

Finding the necessary technologies was not too difficult, because I had used JENA before in Large Scale Databases. Some of the base knowledge of technology was already there to build upon and apply to this project, making it more realistic and applicable to industry.

The research work was reasonably straightforward and investigating research papers was worthwhile. It was difficult to understand some papers as most of the "information extraction" techniques were aimed at Masters and Ph.D. students.

It took extra time to understand the more complicated research articles and as a result, time passed quickly. In future similar projects, more time should be given to research activities. As a task research effort cannot be underestimated and took more time than anticipated.

Use of the agile methodology allowed me to build up very simple prototypes to test different research features. Coding forums provided some intelligence for understanding different extraction and "ontology creation" techniques. This was an effective strategy and achieved innovative results under difficult time constraints.

The requirements period was too short and I only had a few days to capture what the system needed to achieve. The preceding research provided clues as to what the solution would be comprised of.

Designing the software was a very important activity. Having visualised what was required using UML modelling, I proceeded to iterate design and coding tasks. It was a case of taking ideas and simple sketches and using them to build up the formal structures such as the class diagram and sequence diagrams. Once these models were constructed, I felt comfortable with my system architecture and used these important documents to improve the naming conventions, execution order and detail of methods which each function encapsulated.

Implementation work was rewarding and taking an idea such as “using the Levenshtein distance to compare similarity of animal classifications” and then applying this to a given module successfully, encouraged me further. This allowed my creativity to flow and while the solution was more investigative in nature than a standard software development model, an iterative approach was a very good choice for this project.

Testing the system was a long task, which was also performed throughout the project’s iterations. When testing, it was important to be meticulous to ensure robustness and as a result I found many system improvements. Only through testing the more difficult examples did the scope of my system’s functionality become clear. In general, the developed system is very adaptable to simple animals which only have one or two properties which was an acceptable limitation.

I was delighted with progress made throughout the duration of the project. The recursive solution designed for the food-chain hierarchy being a favourite area because of its recursive abilities and programming power to build a large model quickly.

Section 7.2.1 – Project Experience: Positives and Negatives

The following table provides my thoughts on the positive and negative aspects of the project.

Positive Experiences	Negative Experiences
Learning opportunities	Pressure from short timescales
Programming recursive routines	Understanding Ph.D. level material
Reusing code modules effectively	Time taken to document and review the dissertation
Research into modelling	Frustration with finding defects
Iterative development	Poor JENA documentation
Productive meetings with Supervisor	Dissertation formatting
A working system which proved the hypothesis	Updating existing documentation

Section 8 – Conclusions

“Everything that civilisation has to offer is a product of human intelligence; we cannot predict what we might achieve when this intelligence is magnified by the tools that AI may provide...”, (Stephen Hawking, 1st May, 2014).

This research project proved that “*Information Extraction to Infer New Knowledge Using Ontological Modelling*” was feasible. It showed that building a food-chain was a valuable addition to the concept demonstrator.

Different techniques were explored to help achieve better information extraction and ontology creation and were combined into the solution. The developed prototype uses reasoning, inference and assertion over traditional data analysis and Big Data solutions.

The developed prototype is functional, well-documented and in a suitable state for further research that an M.Sc. or Ph.D. programme could provide. By extending the core principles further, a collaboration between industry and academia could lead to the design of a commercial knowledge generation solution, which could take industry into an era of significantly smarter computing.

Applying pattern matching algorithms to relevant webpages provided a good way to extract information. However, patterns need to be apparent from the outset. Finding a way to deal with websites which have a different pattern set was a difficult area for investigation but Natural Language Processing techniques could provide a way to achieve this.

The work explored whether “several words could be searched for in a given piece of extracted text” and utilising Jsoup, it was found that this was possible. However, the context of the word made the challenge difficult.

The webpages used to extract information are key to how useful the analysis will be. The difficult question to answer is how trusted extracted results should be trusted. Some sites often have domain specific knowledge and such “Expert” knowledge could be modelled appropriately with the techniques mentioned.

The modelling of extracted information is relatively straightforward, with much of the functionality coming from JENA. It was a good decision to initially model the ontology in Protégé, to help better visualise it. This ensured that the created ontological model in JENA correctly represented a given scenario (with the correct properties and features being added).

This project has been highly educational and revealed what is available in terms of “information extraction” technologies. JENA’s documentation for ontology building was relatively poor making development more challenging.

The prototype solution was designed to work for the “*animal classification*” domain but with suitable modifications and development, it could be made applicable to many other domain areas. e.g. (1) inferring flight information from and to certain airports,(2) smartphone classification coupled with data mining capabilities, (3) Dinosaurs and Insects to extend the current system. **The extensibility of the program is one of the most useful project outcomes.**

Appendices

Appendix A – Problem Specification Deliverables	52
Table A-1: Requirements Specification & Acceptance Criteria	52
Table A-1 (Continued): Supplementary Requirements.....	55
Appendix B – Design Phase Deliverables.	56
Figure B-1: Prototype Class Diagram	56
Figure B-2: Prototype Sequence Diagram.....	57
Figure B-3: Hierarchy of Supported Animal Classes.....	58
Table B-4: Design Decisions and Rationale Tables	59
Appendix C – Implementation Algorithms.....	64
C: 1 - Information Extraction Pseudocode algorithm	64
C: 2 – OntologyCreation Pseudocode Algorithm	65
Appendix D – Detailed Testing Method & Approach.....	67
D-1: Test Plan	67
Table D-2: Modular/Unit Testing	68
Table D-3: Functional Testing.....	80
Table D-4: System Testing	84
D-5: System Testing Code Output for “Butterfly” and “t-rex”.	85
Appendix E –Screenshots.....	90
Screenshot E-1: Eclipse IDE	90
Screenshot E-2: Eclipse Console and system output	90
Screenshot E-3: Adding dependancies using Maven	91
Screenshot E-4: Corresponding Maven POM.XML file	91
Screenshot E-5: Protégé.....	92
Appendix F – Animal Website Selection Criteria	93

Appendix A – Problem Specification Deliverables

Table A-1: Requirements Specification & Acceptance Criteria

Table Legend: Requirement Codes: F- Functional, D-Design, S-Supplementary

Risk Levels: Nil, Low, Medium, High, Very High

Priority: “Will”= Mandatory, “Should”= Desirable, “Could” = Possible

Requirement ID	Requirement Description	Risk	Acceptance Criteria	Acceptance Method	Notes
F-1	The system will extract HTML data from several different webpages.	Low	Verify that the system has correctly extracted HTML data from 2-3 pages.	Demonstration	Consider adding more webpages for future iterations.
F-2	The system will get useful properties from the extracted HTML data.	Medium	Having analysed the HTML for an animal, the system will return its correct set of properties. e.g. [Lion: hairy, vertebrate... etc.]	Inspection	
F-3	The system will model the extracted properties and animal related data in an ontological form.	Medium	Check that the system has correctly modelled the properties and animals as ObjectProperties and OntClasses respectively.	Inspection	Use “ostrich”, “frog” and “t-rex” for testing.
F-4	The system should be able to generate the ontology itself.	Low	Verify that the system has correctly generated the ontological model by examining system outputs.	Demonstration	
F-5	The system will display suitable error messages for most commonly occurring problems.	Low	Verify that the system outputs are appropriate and indicate where the problem occurred.	Demonstration	e.g. whether URLs are correct or not.
F-6	The system should be able to infer what class most animals belong to.	High	Check that the system can correctly infer what class identified animals belong to.	Inspection	Some animals are difficult to classify and should be accepted as edge-cases.

D-7	The system will be programmed in a well-documented and modular way, using a suitable language.	Low	Check that code has been commented and developed in a modular manner.	Inspection	Java was used.
F-8	The system will return the output of the program to the command line.	Low	Verify that correctly formed ontology information is output.	Demonstration	
F-9	The system should be able to validate itself, comparing extraction results from several webpages and using a “voting system” to determine the likely result.	Medium	Verify that the system has correctly compared the results of extraction and used the “voting system” to determine an appropriate result.	Inspection	Functionality was implemented to standardise the values which were given to the “voting system”.
D-10	The system will have a set of methods dedicated to building the ontological model of animal classifications.	Medium	Verify that the system correctly implements a set of methods to build the ontological model.	Inspection	
D-11	There will functions for testing, with the possibility to extend these further.	Low	Verify that these set of Test functions correctly validate intended behaviour.	Inspection	
F-13	The system will create key value pairs for each animal based on their animal classification.	Medium	Verify that key value pairs have been correctly created for animals and their classifications.	Inspection	

F-14	The system will get the HTML data for one or more different animals to build an overall model.	Low	Verify that the system correctly extracts the HTML data for one or more different animals to build up a model of: Birds, Reptiles, Amphibians, Mammals and Fish.	Inspection	
F-15	The system's ontological model should allow the inference of additional information about animal classifications and associated properties.	Very High	Verify that the system's ontological model allows for additional information inference about animal classifications e.g. "Reptiles always live in a desert".	Inspection.	This requirement was only tagged as " <i>desirable</i> ".
F-16	The system should create a hierarchy based on the "one animal" "eats" "another animal" relationship to create an animal food chain.	Very High	Check that a hierarchy is correctly constructed based on animal predators and prey.	Inspection	
F-17	The system should extend the ontology to include dinosaur classifications and relationships.	Very High	Check whether a correct Ontology of Dinosaur classification and Dinosaurs has been built.	Inspection	
F-18	The system should extend the ontology to include Insect classifications and relationships.	Medium	Verify that insect classifications, properties and associated relationships have been correctly added.	Inspection	
F-19	The system should extend the ontology to include vertebrates and invertebrates.	Low	Verify whether a correct ontology of vertebrate and invertebrate properties has been added.	Inspection	

Table A-1 (Continued): Supplementary Requirements

Req. ID	Requirement Description	Risk	Acceptance Criteria	Acceptance Method	Notes
S-20	The system should generate different lists of animals.	Low	Verify that a list of animals can be correctly generated by the system.	Inspection	Basically allows modifiable input to the program.
S-21	The system should get the HTML for a given page.	Low	Verify that given an input URL, the system will correctly return the HTML for a given page in a "Document" format.	Inspection	
S-22	The system could check the animal classification against expected results	High	Verify that facilities have been provided to check whether an animal has been classified correctly and if not standardise the result.	Inspection	This is to combat user error and is important to check that the animal classification is correct.
S-23	The system could also look for scientific names (usually in Latin) for Animals which can be added to the Ontology.	Low	Verify that scientific names can be correctly added to an Ontological model.	Inspection	
S-24	The system could ask for user input for unclassified results.	Medium	If the class cannot be found verify there is a facility to add manually.	Inspection	

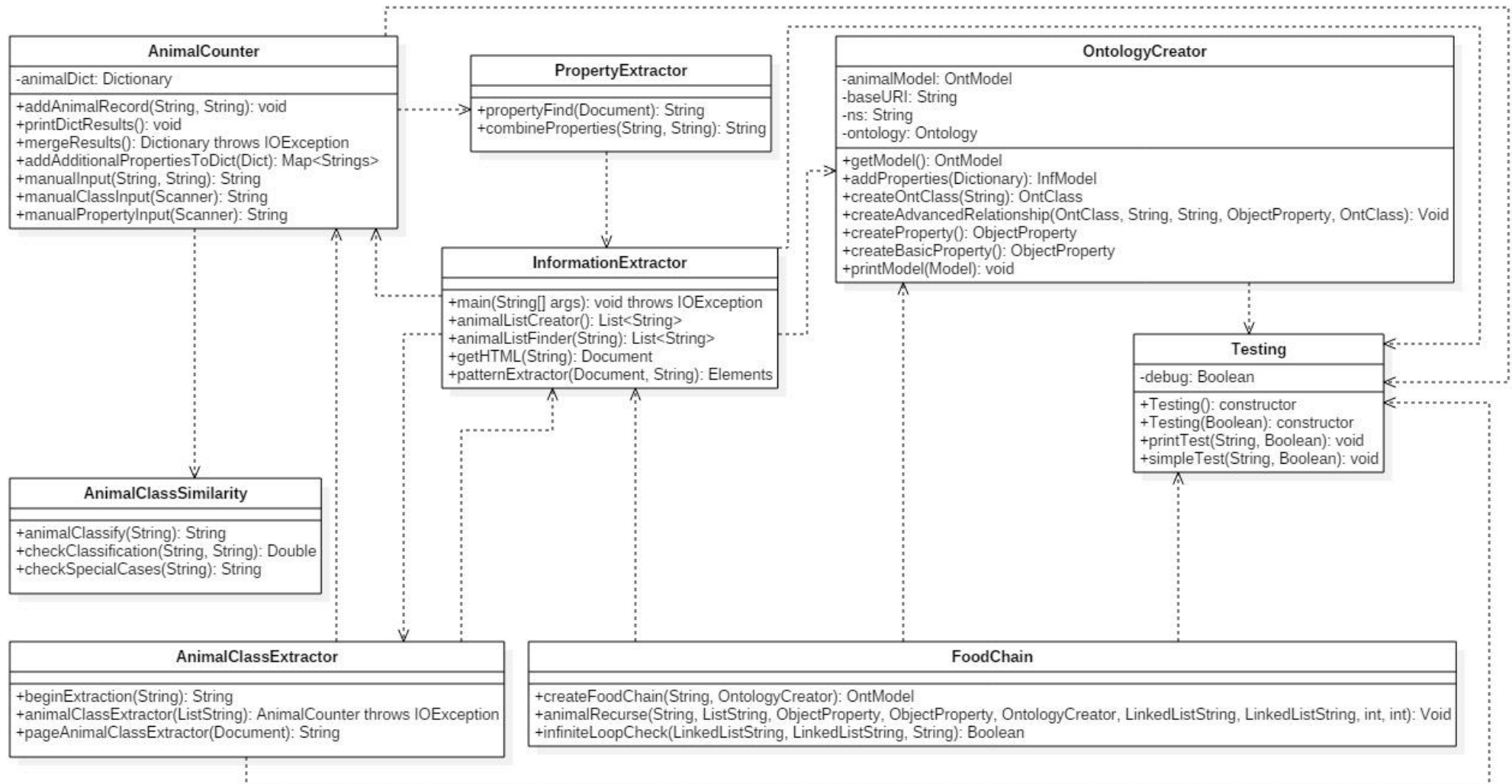
Appendix B – Design Phase Deliverables.**Figure B-1: Prototype Class Diagram**

Figure B-2: Prototype Sequence Diagram

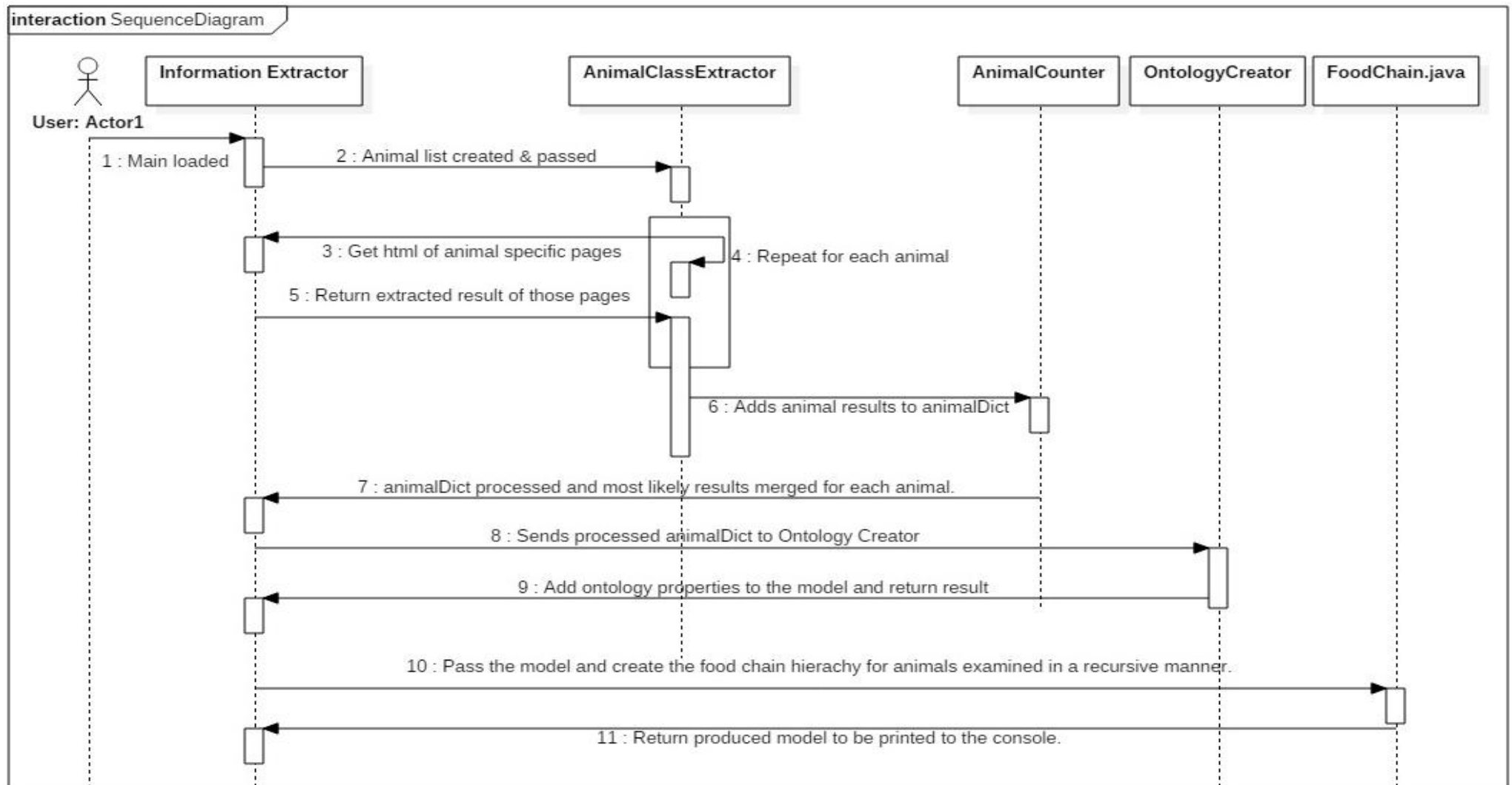


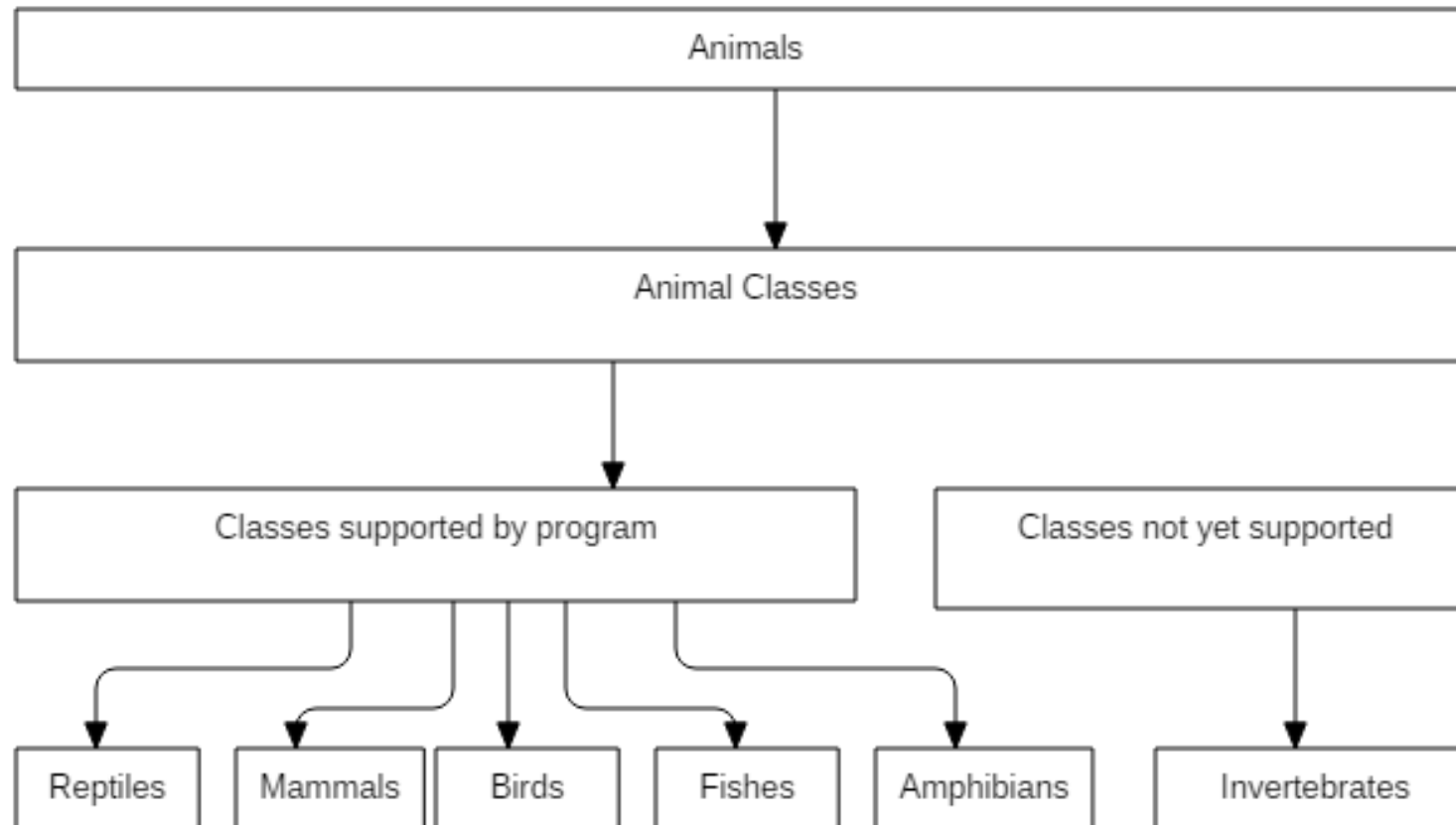
Figure B-3: Hierarchy of Supported Animal Classes

Table B-4: Design Decisions and Rationale Tables

Iteration 1		
Requirement ID & Brief Summary	Design Decision	Rationale
F-1 (Initial), S-21 (Initial) - Extract webpage HTML data for a given animal.	Extract HTML data using the Jsoup method :Jsoup.connect(URL).get();	Jsoup is a well-documented and suitable library for <i>“information extraction”</i> .
F-2 (Initial)- Extract the animal’s classification from website sources.	Extract the animal classification by looking for the text of “Class:” using the “.select()” method of Jsoup , looking within the HTML to find the element which conforms to the criteria and find the text associated with it.	An initial test to learn how easily <i>“information extraction”</i> could be achieved. Later approaches were more focused on extracting properties to infer an animal’s classification.
F-3 (Initial), F-4 (Final) - Generate extracted properties in an appropriate modelled ontology form.	Use of a large method and set of functions to “add” properties to the model and then also print the classification.	This large method was later improved but was added to ensure that appropriate functionality was available.
F-5 (Initial) -Display suitable error messages	A debug condition was created to indicate whether debug output was required.	The plan was to improve upon messaging and error information in each of the subsequent iterations.
D-7 (Initial) -Programmed in a modular, well-documented manner	Java was selected as the programming language.	Java is class based and modular; its libraries are well documented, supported and extensible.
F-8 (Final) - output is returned to the command line.	The output was returned to the command line using the Java print method.	This is a research prototype and does not require a complex GUI.
D-11 (Initial) - Functions for testing	Create initial Java functions to perform different types of output.	These Testing functions would be used at each phase of development to check program correctness.

F-13 (Initial) -Create key value pairs for a given animal and the associated classification.	Created a dictionary object to hold this information.	A dictionary was used because it offered the required functionality to create such pairs.
S-20 (Initial) - Create a list of animals for testing purposes.	Have a String array of arguments which are ready to be passed and then converted to a List<String> object.	This is a simple and pragmatic technique for inputting the same test animals efficiently and easily.

Iteration 2		
Requirement ID & Brief Summary	Design Decision	Rationale
F-1 (Final) -Extract webpage HTML data for a given set of animals.	To extract the HTML data using the same Jsoup method as previously but for several web-pages passed by appending a test URL with an animals String name.	This allowed several animals and their associated HTML to be extracted in one action using a double for-loop, to loop through each test URL link.
F-2 (Initial) - Extract useful properties from the HTML data.	To use Jsoup's ".select" method to find HTML data which matches a given word pattern on a basic level.	It provided all the " <i>information extraction</i> " capabilities required.
F-3 (Intermediate) - Model extracted properties in an ontological form.	To add the properties in a simple, sequential manner while ignoring more complex cases.	Kept simple to allow initial testing to prove that core techniques worked. Ignoring special cases meant that core functionality could be achieved quickly.
F-6 (Initial) - Class inference for a specific animal.	To use the "Pellet" <i>reasoner</i> as the reasoning tool.	"Pellet" was tested with the Protégé tool and it performed the required inference tasks which I wanted to achieve with JENA.
F-9 - Animal Counter object voting system.	To create the AnimalCounter class to emulate the required "voting system".	The voting system allows the validation of the extracted results.
D-10 - (Final) Model extracted properties as an ontology.	To use a set of developed methods in the OntologyCreator.java class to create the ontology and relationships.	This approach produced more modular code and hid away some of the complexity of the solution. It was implemented in different functions rather than in a large block of code (as it was in the previous iteration).

D-11 (Final)- Functions for testing	To improve formatting by adding spaces and lines to separate out content.	Using spaces would make these Testing functions easier to read. This allowed an understanding of when a certain module was being executed.
F-13 (Final) – Create key value pairs for a given animal and their classification.	More complex key value pairs were constructed of both an animal’s classification and their properties.	This was a simple way to use the “split” function to separate out different pieces of information.
F-14 (Final) - Get HTML data for one or more different animals to build an overall model.	To retrieve the HTML for a given page, extracting the properties and performing this process for the required number of classifications.	More classes would provide a better range of animal classifications. Several animals could be featured in a given test.
S-20 (Intermediate) - Create a list of animals for testing purposes.	Building upon the previous design, a Java List Object and the “.add()” function was used to add different test Strings.	This approach is better for adding in test animals because of its simplicity. This is also more extensible.

Iteration 3

Requirement ID & Brief Summary	Design Decision	Rationale
F-2 (Final)- Extract properties in a more sophisticated way.	To use a more specific set of classification words to search and define a property as being present.	A simple way of demonstrating the concept. There is a risk because there are edge cases with properties, such as “feathers”, which are not classified as Birds. This requirement, was more to prove that a refined choice of words could work.
F-3 (Final) - Model these properties in an ontology	To create functionality to add “equivalent” or “restricted” ontology relationships for either an animal classification or animal respectively.	This approach meant that the same code was not being repeated and delivered a more effective function.
F-6 (Final) – Class inference	The inference process was developed further to print the inferred model as well as the original models.	This was a useful feature to check what inference operations had been performed and test whether this functionality was working.

F-9 (Final)- Animal Counter object voting system.	Additional functionality to find the most likely classification for animals based on results which include identified animal classifications as well as unclassified ones, represented by "NONE" values.	It is important to consider other cases where an Animal cannot be classified. This also makes the system more robust as it is able to handle scenarios where classifications are not found. N.B. The system will always default to classified values when the voting process completes.
F-16 (Initial)- Create food chain of animals and the animals they "eat" and are "eatenBy."	To use a recursive function to search through each animal which eats and is eatenBy a given animal.	Recursion was a good choice because this was a tree traversal task which lends itself nicely to a recursive algorithm.
F-19 (Initial) - Invertebrate and vertebrate properties should be considered.	To implement the properties of vertebrates. N.B. Invertebrates were ignored.	The existing property extraction was enhanced with the properties of vertebrates. It was more efficient to use this existing structure rather than creating something from scratch. It also showed the extensibility aspects of the system.
S-20 (Final) - Create a list of animals for testing purposes.	Use a Java List object and the ".add()" function to add test extracted animal Strings from a website of "A-Z" animals.	This is a more thorough testing method and gets a large list of test animals quickly. The code determines which animals should be added based on a modular calculation.
S-22 (Initial)- Checking the animal has an accepted classification.	To implement functionality to check whether the classification is similar enough to an accepted set of classifications.	The well documented Levenshtein distance measure to compare Strings was used.

Iteration 4		
Requirement ID & Brief Summary	Design Decision	Rationale
F-2 (Final), Extract properties in a more sophisticated way.	To build upon the previous work for this requirement to allow for capitalisation to be taken into consideration.	To get a more robust system. A scenario of no properties being extracted was also factored in.

F-16 (Initial) - Create food chain of animals and the animals they “eat” and are “eatenBy.”	To extend the recursive process and make “eats” the inverse property of “eatenBy”.	This approach allowed the system to infer that if a given animal (Lion) is the predator of some animal (Antelope), then an Antelope is the prey of a Lion.
	The tree creation process considered all animals which “eat” and are “eatenBy” in a more advanced manner i.e. traversing all possible outcomes recursively.	This even more recursive approach meant that additional information about animals which was stored on the webpage could also lead to further knowledge inference.
F-17(Initial) – Dinosaurs can be classified in a basic way.	An animal can be stated to belong to the “animal classification” of Dinosaur.	This proved that it was simple to extend the classification system, although the properties for dinosaurs and insects were not considered.
F-18 (Initial) – Insects can be classified in a basic way.	An animal can be stated to belong to the “animal classification” of Insect.	
S-22 (Final) - Checking the animal has an accepted classification.	Taking the previous standardisation functions, also considered cases where animals have a different subclass name.	A dictionary was developed to allow an easy way to map from the value to a key. As long as the input value matches one of the separated Strings in the values part of the dictionary, then that animal could be standardised to be the key of that entry.
S-23 (Final) Allow the scientific names of animals to be added.	If the “ScientificName:” field does exist on a given page, then extract this using Jsoup’s “.select()” method.	This was a test to see how easily new properties could be added.
S-24 (Final) Allow human intervention when a given animal cannot be classified.	If the system cannot retrieve a valid animal classification then the system will ask for user input.	A case statement was used which only would retrieve user input if a user wanted to provide classification information.

Appendix C – Implementation Algorithms.

The following algorithms have been provided to aid in the understanding and replication of this work:

C: 1 - Information Extraction Pseudocode algorithm

As documented previously “mergeResults” function uses a voting system to decide on the most probable classification.

Most of this functionality is largely achieved by the AnimalCounter Java class which adds the probable classification and animal (as the key) and the properties extracted (as the values).

```
Def InformationExtraction (ListAnimals):
```

```
    animalDict = {}
```

```
    For each (animal:ListAnimals):
```

```
        For each link:
```

```
            extractedHTML = retrieveHTML(link + animal)
```

```
            result = extractPatternData(extractedHTML)
```

```
            results += addResultToValuesPartOfDict(result)
```

```
        probableResult = mergeResults(results)
```

```
        properties = extractProperties(animal, extractedHTML) #Also adds the animals ScientificName.
```

```
        put-in-animalDict(probableResult, properties)
```

```
return animalDict
```

C: 2 – OntologyCreation Pseudocode Algorithm

The previous “Information Extraction” process provides the useful input to OntologyCreator.java which can then model the selected animals and their extracted properties.

```
Def addProperties(animalDict):
```

```
    ANIMAL = createOntClass("Animal")
```

```
    For each animalPair in animalDict:
```

```
        If (check-classinfo-added = true):
```

```
            probableClassification= retrieveAnimalClassToBeAdded():
```

```
        Else:
```

```
            probableClassification = animalPair.getKey()
```

```
            acceptableClasses = Array("Mammal","Reptile","Amphibian","Fish","Bird","Unclassified")
```

```
            properties = animalPair.getValues.split(",")
```

```
        If properties.length >= 1:
```

```
            If (check-whether-animalclass = true):
```

```
                animalOntClass = createOntClass(animalKey)
```

```
                For each property in properties:
```

```
                    p = createProperty(property, ANIMAL,ANIMAL)
```

```
add-equivalent-advancedRelationship(animalOntClass,p)
```

```
add-superClass(ANIMAL)
```

```
Else:
```

```
    animalOntClass = createOntClass(animalKey)
```

```
For each property in properties:
```

```
    If (checkForScientificName(property) == true):
```

```
        property = extractScientificName
```

```
    p = createProperty(property, ANIMAL,ANIMAL)
```

```
    add-restricted-advancedRelationship(animalOntClass,p)
```

```
    add-superClass(ANIMAL)
```

Appendix D – Detailed Testing Method & Approach

D-1: Test Plan

The approach for testing this prototype solution was focused on checking whether the requirements had been satisfied for each development iterations. Code correctness was key and checks were made whether the solution was performing as expected against a number of test inputs. The testing processes were repeated whenever changes were implemented

- **Module/Unit Testing**

- Each module was individually tested to check that it was working as expected. Program outputs for various inputs were documented in advance and the results obtained were compared.

- **Integration Testing**

- Combining modules which had already been unit tested and checking that they still work as expected as an integrated whole. This involved taking the modules which handled initial execution and slowly building up the overall program in each iteration until it performed more and more prioritised functionality as further modules were added.
- This approach should reveal problems to do with the way variables are being passed and returned and has been performed throughout the project during each iteration.

- **Functional Testing**

- Checking that the iteration requirements had been satisfied and that the system was robust enough to be used as a prototype solution.
- Evaluation criteria which were specified at the problem specification phase were used to evaluate how close the prototype was to the desired specification.

- **Systems Testing**

- The system was tested as a whole via "Black Box Testing" methods, where program outputs were compared to expected results for given inputs.

Table D-2: Modular/Unit Testing

Feature being tested	Why it being tested?	How is it being tested?	Expected Result	Pass? Y/N	Test Comments.
<u>InformationExtraction.Java</u>					
[I1]-Animal List Creation	Future tests rely on this component.	Adding animals from a String[] to the list and printing the result to check they are being added properly.	All animals in the appended list should be output.	Yes	animalList is a String[] and converted to a list of Animals which is then returned.
[I2]-Animal List Creation	Future tests rely on this component.	Adding animals to the animalList using the .add() method of a List<String> to check they are being added properly.	All animals in the appended list should be output.	Yes	animalList is a List<String> and animals are added with the .add() method. List.add("Lion") as an example.
[I3]-Animal List Creation	Future tests rely on this component.	Adding animals to the animalList using the Arrays.asList(String[] animals) method to check they are being added properly.	All animals in the appended list should be output.	Yes	animalList is appended by the animal array which is a parameter in the Arrays.asList() method.
[I1]-Logging Capabilities Enabled	Removes error messages from the console.	Examining the output to check that error messages are not present.	No errors should be reported relating to the logging status.	Yes	By importing and using the Log4j basic configurator unnecessary messages are removed and logging facilities are available.
[I1]-Extract HTML Page Content for a given URL.	Check that Jsoup is providing the HTML extraction facilities correctly.	Testing that when the function is run that it returns correct HTML for a given URL and page.	Function returns the HTML associated with a given page.	Yes	The Jsoup extraction method gets all the HTML associated with a page. Later processing extracts the specific details. Testpage: http://a-z-animals.com/animals/lion/
[I1]-Extract HTML Page Content for a given URL.	Check that incorrectly typed URLs are trapped	Inputting an invalid URL into the program	The URL should cause the program to throw an exception.	Yes	The invalid URL I used was a http://a-z-anmals.com/ This would be a simple case of mistyping the address.

	and do not generate errors.				
[I1]Pattern Extraction	Check that the returned result is correct.	Take an extracted Document page and select the parts of the HTML which match the given pattern.	That returned result matches what is seen in the HTML code.	Yes	The Jsoup method “.select()” can be used to extract information based on the pattern.
[I2]Pattern Extraction	Check that the returned result is correct.	Take a Document page and select the parts of the HTML which do not have the given pattern.	That the returned result is null.	Yes	The method is syntax specific so when no result is returned it is usually due to a small syntax change being required.
[I1]-Overall module	Check that all features of the module work together	Take a set of animals, get the HTML for each of them based on a startUrl and append the animal’s name. Apply a pattern to search for something unique to each.	That no errors result. I will be testing animals “Lion, Crocodile, Frog and Wild boar”. (This is an edge case to challenge the program)	No	The reason this test failed was because “Wild boar” would not process into a valid URL. I.e Only “Wild_boar” is acceptable.
<u>AnimalCounter.Java</u>					
Feature being tested	Why it being tested?	How is it being tested?	Expected Result	Did it pass the test	Test Comments.
[I1]-Add animal and a property to the internal dictionary.	Ensure that an animal(key) and its property(values) can be added to the dictionary.	Create an AnimalCounter object and add an animal and a property using the “addAnimalRecord()” method.	Animal should be added as the key and the values should be equal to the property set.	Yes	Animal Counter is its own data structure which is a bit more complex than a Dictionary.
[I2]-Add animal and its properties to the internal dictionary.	Ensure that an animal(key) and its properties(values) can be added to the dictionary.	Create an AnimalCounter object and add an animal and a property using the “addAnimalRecord()” method. Adding a second property using the same method.	Animal should be added as the key and the values should be equal to the properties set.	Yes	When testing 5 properties were correctly added in the same way for a given animal.

[I1]-Print results contained in the AnimalCounter object.	Check whether the print function is working.	Examining the output from the animalcounter.printDictResults() method.	The key should be printed with a ":" and the values should be printed after.	Yes	Provides an easy way to print the dictionary to standard output.
[I1]-Merge together results of an AnimalCounter object.	Functionality to combine results from several sources.	Merge a list which contains an animal and 2 of the same classifications. E.g [Lion: Mammal,Mammal]	Should return the animal with just the most frequently occurring classification -> i.e. [Lion:Mammal]	Yes	
[I2]-Merge results of an animalCounter object.	Functionality to take results from several sources and combine them.	Merge a list which contains 2 similar classifications and 1 different classification. e.g. [Lion: Mammal,Mammal,Bird]	Should still return the animal with the most occurring classification. i.e. [Lion:Mammal]	Yes	
[I3]-Merge results of an AnimalCounter object.	Functionality to take results from several sources and combine them.	Merge a more complex list of animal and classifications e.g. [Lion:Mammal,Mammalia,Bird]	Return the animal with the most frequently occurring classification. i.e. [Lion:Mammal]		This was completed during the final iterations. Use of the animalClassify() method to convert classifications to a simple and accepted form.
[I3]-Merge results of an animalCounter object.	Functionality to take results from several sources and combine them.	A long list of challenging dictionary elements would be input and checked that they are being merged correctly.	Should still return the animals each with their most occurring classification.	Yes	
[I3]-Add a set of additional properties to the already classified AnimalCounter.	Properties provide the inference capabilities for the JENA reasoner.	Examining the dictionary output to ensure that properties are being added correctly.	That the dictionary adds the expected properties into the values and adds the correct key for that animal.	Yes	Later iterations began to examine "properties" more closely than extracting the classification which meant more of an inference approach was being applied.

[I3]-Checking whether a result for the animal's classification has been worked out successfully.	Provide the option for human intervention where an animal's classification cannot be found.	Using as input an animal which does not possess an animal classification in the web sources.	The system should ask whether the user wants to add their own classification.	Yes	If there is no "largest key count" i.e no animal classification result: The system asks "Do you know the classification(Y/N)?"
[I4]-Checking whether a result for the animal's classification has been worked out successfully.	Provide the option for human intervention when an animal's classification cannot be found.	If a user knows the classification the system should handle this appropriately.	When a user states they know an animal's classification and inputs ("Y") or ("y") the system should ask for the manual animal classification and check this is valid.	No	A case conversion function was used to get the lowercase value of "y" to work as expected. The system was parsing an exact input of "Y" only in the first instance.
[I4]-Checking whether a result for the animal's classification has been worked out successfully.	Provide the option for human intervention when an animal's classification cannot be found.	If a user does not know the classification the system should handle this appropriately. Note: if the user types anything other than ("Y","y","N","n") the system will handle that input as "no" or "n".	When a user states they do not know an animal's classification and inputs ("N") or ("n") the system should not ask for any further input for that animal.	No	See previous comment.
[I4]-Checking whether a result for the animal's classification has been worked out successfully.	Provide the option for human intervention when an animal cannot be added.	Given that an animal currently possesses no classification and a user knows what the classification is (they input "Y" previously).	The system should allow the user to add a new classification if they want to and this value will be added as the new largestKeyCount.	Yes	The system will then prompt for the input from the user and if this is an accepted animal class it will add the classification to that animal's entry in the dictionary.

[I4]-Checking whether a result for the animal's classification has been worked out successfully.	In the case of an invalid input, don't add anything to the animal entry in question.	Trying to manually add a classification which is not yet supported e.g. "Insect" If a user types anything else other than a specified animal classification ("Bird,"Mammal", etc) the input is not added.	The system will tell the user they have tried to add an incorrect classification and tells them that this input has not been added.	Yes	Also outputs the set of accepted animals.
<u>AnimalClassExtractor.Java</u>					
Feature being tested	Why it being tested?	How is it being tested?	Expected Result	Did it pass the test	Test Comments.
[I2]-URLs being used for Information Extraction	Check that the web source of : http://a-z-animals.com/ is classifying properly	Compare the results of extraction with the results of manually looking at the animal's classification for a given set of animals.	The results of extraction for a given animal for this URL should match the results found when manually looking at the same content on the actual webpage.	Yes	The only problem is when an animal does not have valid webpage content.
[I3]-Test the more difficult cases of classification	Whether the system can handle more complex cases.	Adding a difficult set of animals as an input to this class. e.g ["seaturtle","flyingfox"]	That the system can classify some results but not all.	No	I found that I needed to add an underscore to animals which have several words which make up their name.
[I4]-Test the more difficult cases of classification	Whether the system can handle impossible cases of classification.	Adding a set of animals which are not animals! e.g. "Dragon", "Cyclops"	That the system does not crash when these results are given as an input.	Yes	These results simply return no classification. However, "Dragon" does have properties such as "wings" which rather interestingly can be used to infer that a "Dragon" is a Bird, although this is not quite accurate.

[I2]-The result which is calculated is added to the dictionary.	Ensure that that results are being added correctly into the dictionary.	Providing a set of animals and web page source links and ensuring where animal classification has been successfully achieved that these results are added correctly to the dictionary.	The system adds an animal and its classifications to the dictionary structure ready to be “merged”.	Yes	Results are added to the dictionary with the key being the animal and its values being the classifications which have been determined.
[I2]-The result which is calculated is added to the dictionary.	Ensure that results are being added correctly into the dictionary.	Providing a set of animals and web page source links which have a mix of both classified and unclassified results and checking both of these results are being added correctly.	The system adds an animal and its classifications as well as the “NONE” classifications to the dictionary structure ready to be “merged”.	Yes	This is so that the AnimalCounter object only has to deal with “merging” results which are a part of an accepted set of classifications and therefore this ignores any results which are unclassifiable.
[I3]-Where no result is calculated that this result is added appropriately.	Ensure that where no result is calculated that this is added appropriately.	Add an animal which cannot be classified as input.	The system adds this animal and its classifications of “NONE” to the dictionary structure ready to be “merged”.	Yes	Where no results are found the system will append “NONE” to the same dictionary structure. However the “merge” operation ignores this phrase, thus it just indicates that no result was found.
<u>PropertyExtractor.Java</u>					
Feature being tested	Why it being tested?	How is it being tested?	Expected Result	Did it pass the test	Test Comments.
[I2]-Property Extraction	Core functionality.	Extracting properties for a easy to classify animal. E.g. Ostrich.	That the property which is added is “wings”.	Yes	Birds are a good choice for a simple test, finding the word “wings” or “feathers” is almost always true on a website involving Birds.

[I3]-Property Extraction	Core functionality.	Extracting properties for more difficult supported case E.g. barracuda	That the property which is added is "gills".	Yes	Fish are a lot more difficult to classify and its only by virtue that they relate to a superclass of "Fish" which classifies them on the "a-z animals" website used for property extraction
[I3]-Property Extraction (when no properties are found for a given animal)	There are circumstances where the HTML which is being returned is relating to the "general" A-Z result page which catches incorrect URL inputs.	Trying to extract the properties for an animal which is not stored on the site in question of "a-z animals.com" e.g Silver Arrowana	That the system gets no properties and appends "NOPROPERTIESFOUND" to the unclassified animal's entry.	Yes	The system checks whether the animal specified is contained in the page and if it is not it infers that therefore the page does not have the relevant information.
[I3]-Property Extraction (when no properties are found for a given animal)	The problem here is when the URL does have the correct name in it but does not contain any of the required information.	Trying to extract the properties for an animal does not contain any of the properties which are featured. e.g. old_english_sheepdog	That the system gets no properties and appends "NOPROPERTIESFOUND" to the unclassified animal's entry.	Yes	The system checks for these properties and if none are found it adds the appropriate statement.
<u>AnimalClassSimilarity.Java</u>					
Feature being tested	Why it being tested?	How is it being tested?	Expected Result	Did it pass the test	Test Comments.
[I3]-Standardising animal classifications	Core functionality.	Giving an input of "eptile" and seeing whether the class in question can handle this sort of error.	The standardised word returned will be "Reptile".	Yes	As long as the input word matches one of the set of "acceptable" animal classifications by at least 50% then that animal classification will be returned.

[I3]- Standardising animal classifications	Additional functionality added which needs to be tested.	Given an input of an animal which can be standardised, the system should return the standardised form.	The standardised word should be returned e.g. Actinopterygii - >returns Fish	Yes	Another type of fish was commonly called "Chondrithyes" or cartilaginous fishes. I added this and other subsequent classifications, retested and these changes did not affect the programs execution.
[I4]- Standardising animal classifications	Handle cases where the calculated animal classification is not found in the set of "acceptable animal classes" and cannot be standardised.	Adding an animal such as "T-rex" which cannot be standardised without the new acceptable animal class of Dinosaur being added.	T-rex is classified as a Dinosaur but no acceptable animal class is currently available.	Yes	Dinosaurs were later added to the set of acceptable animal classes and now the program can classify Dinosaurs on the basic level. Additionally, I found that "crustacean" animals could be extracted and were added to the standardising dictionary so they would become Fish.
<u>OntologyCreator.Java</u>					
Feature being tested	Why it being tested?	How is it being tested?	Expected Result	Did it pass the test	Test Comments.
[I2]-Adding a set of animal classification properties	The animal's classification properties need to be added as an "equivalent" relationship.	Providing as input an animal classification's properties e.g. ["Bird", "wings"]	Results are added to the ontological model as an "equivalent" to relationship.	Yes	This works for any animal classification which is specified in the animalClassesArray.
[I2]-Adding a set of animal properties	The animal's properties need to be added as a "restriction" relationship.	Providing as input an animal's properties e.g. ["Ostrich", "wings"]	Results are added to the ontological model as a "restriction" on relationship.	Yes	

[I3]-Adding a set of animal properties	In the case of no properties being found.	Providing as input an animal which will not have properties extracted. e.g ["Yorkshire_terrier"]	The class is created but no properties are added.	Yes	
[I1]-Ontology Creation	Core functionality.	Is an animal ontology and their relationships created?	Ontology is created with animals and their classifications.	Yes	
[I2]-Model Inference.	This allows the inference of unknown knowledge to be retrieved.	Giving as input an ontology which has the potential to provide inference.	The class should create an ontology of inferred animals and their related properties.	Yes	The "Pellet" reasoner was used to achieve this.
<u>OntologyCreator.Java - Later Iterations</u>					
Feature being tested	Why it being tested?	How is it being tested?	Expected Result	Did it pass the test	Test Comments.
[I3]-Complex functionality to add properties while classification information is still present.	More complex functionality to get the properties via splitting by the "#". This means the animals classification is still present to check when testing.	Adding a set of animal properties which have also been appended by an animal classification. ["Ostrich","Bird#wings"]	The OntClass Ostrich has an ObjectProperty of "wings" added as a "restriction" on its subclass relationship.	Yes	

[I4]-Complex functionality to add properties while classification information is still present.	As above	Adding a set of animal properties which have also been appended by an animal classification. ["Ostrich#Bird","wings"]	The OntClass Ostrich has ObjectProperty of "wings" added as a "restriction" on subclass relationship.	Yes	Realised that it made more sense to include the animal classification in the key. Ostrich#Bird makes the property splitting process sensible.
[I4]-Scientific name property being added	The scientific name for some animals is an interesting property to examine.	Inputting an animal which has a scientific name.	The scientific name should be added as a property for that animal.	Yes	A Lion should have a scientificName property of Panthera Leo.
[I4]-Scientific name property being added	Some animals do not have a scientific name in the website being searched.	Inputting an animal without a record for a scientific name.	The scientificName property should not be added.	Yes	Uses "if" logic conditions to determine whether the scientificName property has been set.
<u>Testing.Java</u>					
Feature being tested	Why it being tested?	How is it being tested?	Expected Result	Did it pass the test	Test Comments.
[I1]-Appropriate testing functionality is present.	Core functionality.	Code inspection to check for testing methods.	Testing methods are implemented and they offer useful Testing functionality.	Yes	

[I1]- Debugging	The value of the “debug” switch should set whether functionality is performed.	Check whether these debugging functions can be switched off.	Debugging functions offer the facility to be turned off and on.	Yes	
[I1]-Input to test functions	The input to the test function should be correctly printed.	Adding different String inputs to the test functions.	Checking that they are performing required functionality	Yes	
<u>FoodChain.Java</u>					
Feature being tested	Why it being tested?	How is it being tested?	Expected Result	Did it pass the test	Test Comments.
[I4]- Searching for a given animal’s predators and prey.	So animals which have spaces in the name do not cause errors.	Checking that animals with spaces do not cause errors.	That the system runs through these tests fine.	No	An “_” replaced a space in a name to fix the problem.
[I4]-Ontology properties added correctly.	The “eatenBy” and “eats” properties should be correctly added to the ontology.	Checking whether the ontology contains the correct property added for predators/prey of a seed animal.	The predators/prey should be successfully added.	Yes	Tested for one animal initially and that the properties were added correctly. Recursive techniques were then employed.
[I4]-Recursive search for predators and prey of animals.	Check that different animals can still be recursively searched.	Giving an input of “crocodile” and checking whether appropriate properties are added to an Ontology	That properties are added for a “crocodile.” (Or a different animal)	Yes	This system retrieved most of the necessary animals and their relationships, but problems with infinite recursion. E.g. Lion eats Human, Human eats Lion.

[I4]- Add to the list of animals added so far.	To prevent an infinite loop from occurring.	Examining the output and checking whether the list of animals was added.	That every animal which is recursively searched through is added to the list.	Yes.	The list is populated correctly.
[I4]- Infinite loop prevention.	Prevent an infinite loop from a 2 way "eats" or "eatenBy" relationship.	Using an animal which makes an infinite loop occur and checking whether the system responds appropriately.	That the infinite loop is prevented.	Yes	

Table D-3: Functional Testing

Requirement No.	Requirement Description	How is it being tested?	Did it pass the test	Test Comments.
F-1	The system extracts HTML data from several different webpages.	By extracting HTML data from: a-animals, Wikipedia and sandiegozoo.	Yes	The AnimalPropertyExtractor.java class was tested in Modular/Unit Testing.
F-2	The system will get useful properties from extracted HTML data.	The PropertyExtractor.java class implemented this desired behaviour. It was verified that appropriate properties had been retrieved.	Yes	As above.
F-3	The system will model the extracted properties and animal data in an ontological form.	The OntologyCreator.java class implements these two requirements and animal properties extracted by the "InformationExtraction" process are modelled correctly. The model can then be subject to a <i>reasoner</i> such as "Pellet".	Yes	<p>"Information extraction" provides the dictionary of animals and their animal classification and properties extracted and OntologyCreator.java then takes the properties extracted and adds them to the ontological model for a given animal.</p> <p>The OntologyCreator.java class was tested previously in Modular/Unit testing.</p>
F-4	The system should be able to generate the ontology itself.			
F-5	The system will display suitable error messages for most problems.	The most obvious problems/error situations were tested and suitable error messages were generated.	Yes	E.g. an error to do with getting the HTML for a URL which does not have a valid webpage on the internet.
F-6	The system should be able to infer what class most animals belong to.	The system performs the inference process through the OntologyCreator.java class which contains a method to printModel() which uses the "Pellet" reasoner to add assertions.	Yes	It is difficult to perform inference for "Amphibians" and "Mammals" due to the lack of relevant information which is found in the webpages.

D-7	The system will be programmed in a well-documented and modular way. Using a language which is suitable for this purpose.	Java was used for this reason.	Yes	The code is well documented.
F-8	The system will return the output of the program to the command line.	Output from system functionality (e.g. the Ontology) is presented to the user.	Yes	
F-9	The system should be able to validate itself; comparing results of extraction from several webpages and using a “voting system” to find the likely classification.	The AnimalCounter.java class was created for this purpose and the “merges” operation performed this “voting system”.	Yes	The AnimalCounter.java class was tested in Modular/Unit Testing.
D-10	The program will have a set of methods dedicated to building the ontological model of animal classifications.	These functions are implemented in OntologyCreator.java and were tested previously.	Yes	The OntologyCreator.java class was tested in Modular/Unit Testing.
D-11	There will be a set of functions for testing, with the possibility to extend these further.	The Testing functions are implemented in Testing.java and were tested previously.	Yes	The Testing.java class was tested in Modular/Unit Testing.
F-12	The system will output the created ontology to the command line.	The OntologyCreator.java class contains printModel() which prints the model before and after inference has occurred.	Yes	The OntologyCreator.java class was tested in Modular/Unit Testing.

F-13	The system will output key value pairs for each animal based on their animal classification.	Verify that the AnimalCounter.java class performed the “merges” operation to determine the most likely animal classification.	Yes	The AnimalCounter.java class was tested in Modular/Unit Testing.
F-14	The system will get the HTML data for several different animals to build an overall model.	Verify whether the system can implement this requirement given that several animals are used as input.	Yes	Could be only testing with one animal.
F-15	The system’s ontological model should allow the inference of additional information about animal classifications and associated properties.	Verify that the system’s ontological model allows for additional information inference about animal classifications e.g. “Reptiles always live in a desert”.	No	This is yet to be implemented.
F-16	The system should create a hierarchy based on the “one animal” “eats” “another animal” relationship to create an animal food chain.	Check that a hierarchy is correctly constructed based on animals and their predators and prey.	Yes	
F-17	The system should extend the ontology to include dinosaur classifications and relationships.	Check whether a correct Ontology of Dinosaur/Insect classification has been built.	Yes	A very simple ontology can be created of a dinosaur/insect.
F-18	The system should extend the ontology to include Insect classifications and relationships.			
F-19	The system should extend the ontology to include vertebrates and invertebrates.	Verify that vertebrate/invertebrate classifications, properties and associated relationships have been correctly added.	Yes	Invertebrates were not implemented.
S-20	The system should generate different lists of animals.	Verify that a list of animals can be correctly generated by the system.	Yes	More sophisticated approaches were later developed and all passed testing.

S-21	The system should get the HTML for a given page.	Verify that given an input URL, the system will correctly return the HTML for a given page in a "Document" format.	Yes	This supplementary requirement was added due to the way Jsoup worked. It was tested throughout and did not fail without an incorrect URL.
S-22	The system could check the animal classification against expected results	Verify that facilities have been provided to check whether an animal has been classified correctly and if not standardise the result.	Yes	For [Snake,Eptile] the system should return [Snake,Reptile].
S-23	The system could also look for scientific names (usually in Latin) for Animals which can be added to the Ontology.	Verify that scientific names can be correctly added to an ontological model.	Yes	This was implemented and tested in iteration 4 in "OntologyCreator.java". e.g Lion hasScientificName Panthera Leo
S-24	The system could ask for user input for unclassified results.	If neither class nor property can be found these can be added manually.	Yes	If a user incorrectly classifies an animal then this is a problem.

Table D-4: System Testing

Feature being tested	Why it being tested?	How is it being tested?	Expected Result	Did it pass the test?	Test Comments.
Checking code output is sensible	To improve test readability.	Looking at the output and checking that it is readable.	The output should be easy to read.	Yes	Formatting changes were made to make content easier to read and print then it passed.
Unexpected results (such as "T-Rex") do not crash the program.	The program should be robust.	2 difficult to handle cases were used: Butterfly and T-Rex	The program should not crash but handle test cases gracefully.	Yes	The system did not crash and obtained properties of "hairy,wings" for the Butterfly page.
System runs in an acceptable timeframe.	The system should not take longer than it would take to manually check results.	Timing the programs execution for classifying 10 animals.	The system should return the results in less a minute.	Yes	"Old_English_Sheepdog,NONE,NONE,NONE" was a problem in the merge() function. This was later rectified and then the program executed in an acceptable time.

D-5: Unit Testing Code Output for Lion

The calculated dictionary entries for a Lion provided a good way to test whether the system was adding properties correctly and whether or not the voting system was working. If the Lion has the property of “hairy” then I know that the correct property has been added because it is a “Mammal”. The system should also add this classification data to the key. Both of these features were present as can be shown below.

Passing list of animals & getting properties for each animal:

Animal:Lion

<https://en.wikipedia.org/wiki/Lion> classified: Mammal

<http://a-z-animals.com/animals/Lion> classified: Mammalia

<http://animals.sandiegozoo.org/animals/Lion> classified: Mammalia (Mammals)

Lion

Mammal

Animal:Lion

Lion-Class to be added:Mammal

Scientific name cannot be found:<http://animals.sandiegozoo.org/animals/Lion>

Scientific name added:Panthera leo from source:<http://a-z-animals.com/animals/Lion/>

Animal Dictionary contains

{Lion#Mammal=hairy,hasScientificName%Panthera leo}

Final animal dictionary of properties: {**Mammal#ANIMALCLASS=hairy**, Fish#ANIMALCLASS=gills, Snake#Reptile=scaled, Amphibian#ANIMALCLASS=coldBlooded,eggs, Unclassified#ANIMALCLASS=NOPROPERTIESFOUND, poison_frog=eggs,coldBlooded, Reptile#ANIMALCLASS=scaled, Bird#ANIMALCLASS=winged, **Lion#Mammal=hairy,hasScientificName%Panthera leo**}

This result can then be passed to the “OntologyCreator.java” function to create a model of these animals and their properties.

D-6: System Testing Code Output for “Butterfly” and “t-rex”.

Program output was examined to check that this system test had passed. This example additionally illustrates the way Modular and Unit testing was performed with relevant test messages showing when the system cannot perform extraction on a given page.

Butterfly is not a mammal as the system infers, but the edge-case “butterflies” are a hairy Insect, such Insects are out of scope for this project. Dinosaur properties are also out of scope and this is why the system does not extract any suitable properties.

The system does not fail to execute in either of these situations, ensuring even out of scope “animals” do not crash the system.

```
Start main
*****

-----
Animal List chosen:[butterfly, t-rex]
*****

-----
Passing list of animals & getting properties for each animal:
*****

Animal:Butterfly
https://en.wikipedia.org/wiki/Butterfly classified: Insect
```

```
http://a-z-animals.com/animals/Butterfly classified: Insecta
http://animals.sandiegozoo.org/animals/Butterfly classified: Insecta (Insects)

Animal:T-rex
https://en.wikipedia.org/wiki/T-rex classified: Dinosaur
JSoup URL Error: Page could not be accessed : http://animals.sandiegozoo.org/animals/T-rex
Butterfly

Insect
Animal:Butterfly
Butterfly-Class to be added:Insect
Scientific name cannot be found:http://animals.sandiegozoo.org/animals/Butterfly
Scientific name added:Papilionoidea from source:http://a-z-animals.com/animals/Butterfly/
T-rex

Dinosaur
Animal:T-rex
T-rex-Class to be added:Dinosaur
JSoup URL Error: Page could not be accessed : http://animals.sandiegozoo.org/animals/T-rex
Scientific name cannot be found:http://www.example.com/seanjames/document
Scientific name cannot be found:http://a-z-animals.com/animals/T-rex/

T-rex cannot be classified by system

Do you know the properties(Y/N)
n
Animal will not be classified
```

Add properties for T-rex and Butterfly only (Ignoring other unimportant dictionary entries).

```
Animal Dictionary contains
{Butterfly#Insect=hairy,winged,hasScientificName%Papilionoidea, T-rex#Dinosaur=NOPROPERTIESFOUND}

-----
Create ontology of animals and their properties:
*****
{#Insect=hairy,winged,hasScientificName%Papilionoidea, T-rex#Dinosaur=NOPROPERTIESFOUND }

Properties for:Butterfly:[hairy, winged, hasScientificName%Papilionoidea]
```

Should be classified as:Insect
Animal being added:Butterfly
Property being added:hairy
Property being added:winged
Property being added:hasScientificName%Papilionoidea
Split result[hasScientificName, Papilionoidea]

Properties for:T-rex:[NOPROPERTIESFOUND]
Should be classified as:Dinosaur
Animal being added:T-rex
Property being added:NOPROPERTIESFOUND

Ontology (With only Insect and Dinosaur classes represented)

```
<http://www.example.com/seanjames/ontologies/animals#Butterfly>
  a      owl:Class ;
  rdfs:subClassOf <http://www.example.com/seanjames/ontologies/animals#Animal> ;
  rdfs:subClassOf
    [ a      owl:Restriction ;
      owl:onProperty <http://www.example.com/seanjames/ontologies/animals/winged> ;
      owl:someValuesFrom <http://www.example.com/seanjames/ontologies/animals#Animal>
    ] ;
  rdfs:subClassOf
    [ a      owl:Restriction ;
      owl:onProperty <http://www.example.com/seanjames/ontologies/animals/hairy> ;
      owl:someValuesFrom <http://www.example.com/seanjames/ontologies/animals#Animal>
    ] ;
  <http://www.example.com/seanjames/ontologies/animals/#hasScientificName>
    <http://www.example.com/seanjames/ontologies/animals#Papilionoidea> ,
<http://www.example.com/seanjames/ontologies/animals#hasScientificName> .
```

```
<http://www.example.com/seanjames/ontologies/animals#T-rex>
  a      owl:Class ;
  rdfs:subClassOf <http://www.example.com/seanjames/ontologies/animals#Animal> ;
  rdfs:subClassOf
```

```
[ a      owl:Restriction ;
  owl:onProperty <http://www.example.com/seanjames/ontologies/animals/NOPROPERTIESFOUND> ;
  owl:someValuesFrom <http://www.example.com/seanjames/ontologies/animals#Animal>
] .
```

.....

Inferred model:

```
<http://www.example.com/seanjames/ontologies/animals#Butterfly>
  a      owl:Thing , <http://www.example.com/seanjames/ontologies/animals#Animal> , owl:Class ;
  rdfs:subClassOf owl:Thing , <http://www.example.com/seanjames/ontologies/animals#Bird> ,
<http://www.example.com/seanjames/ontologies/animals#Animal> ,
<http://www.example.com/seanjames/ontologies/animals#Mammal> , _:b1 , _:b2 ;
  <http://www.example.com/seanjames/ontologies/animals/#hasScientificName>
    <http://www.example.com/seanjames/ontologies/animals#Papilionoidea> ,
<http://www.example.com/seanjames/ontologies/animals#hasScientificName> ;
  owl:disjointWith owl:Nothing ;
  owl:equivalentClass <http://www.example.com/seanjames/ontologies/animals#Butterfly> ;
  owl:sameAs <http://www.example.com/seanjames/ontologies/animals#Butterfly> .

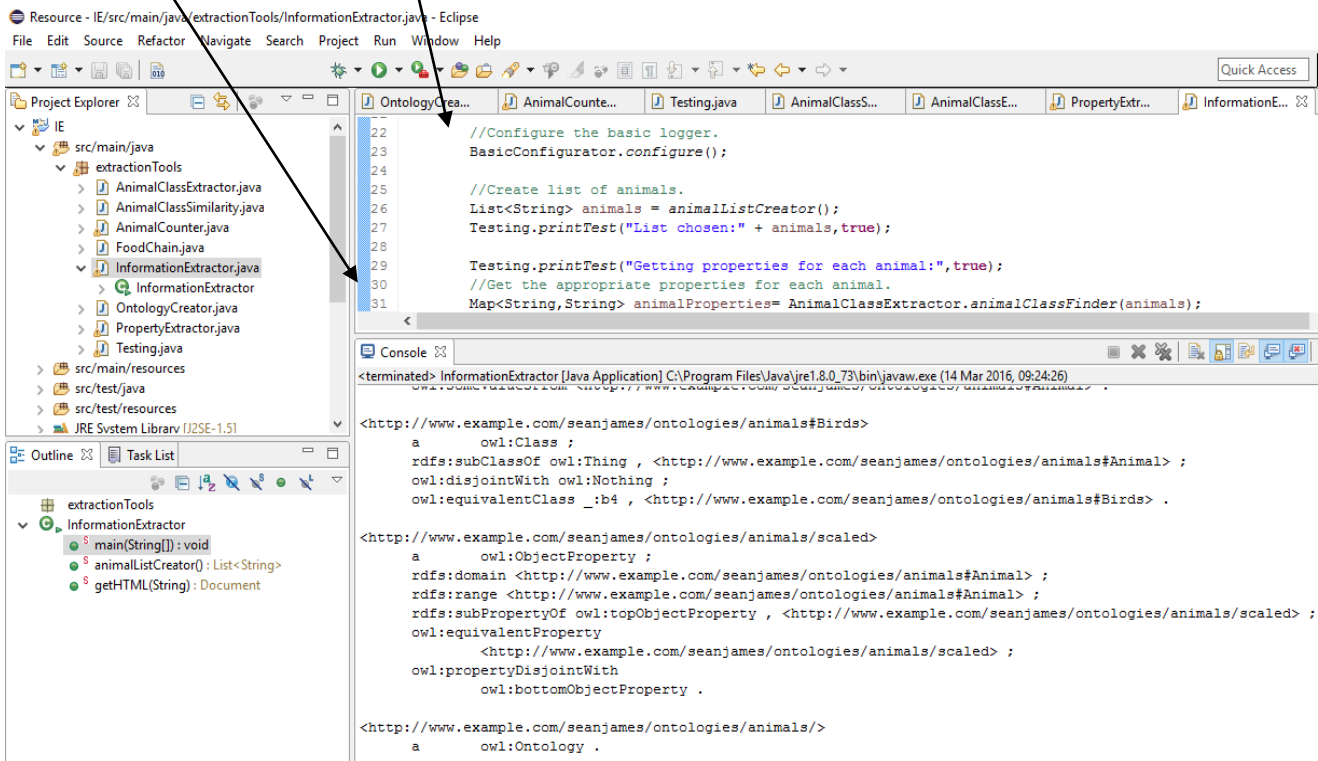
<http://www.example.com/seanjames/ontologies/animals#T-rex>
  a      owl:Class ;
  rdfs:subClassOf owl:Thing , <http://www.example.com/seanjames/ontologies/animals#Animal> ,
<http://www.example.com/seanjames/ontologies/animals#Unclassified> , _:b12 ;
  owl:disjointWith owl:Nothing ;
  owl:equivalentClass <http://www.example.com/seanjames/ontologies/animals#T-rex> .

<http://www.example.com/seanjames/ontologies/animals#Papilionoidea>
  a      owl:Thing , <http://www.example.com/seanjames/ontologies/animals#Animal> , owl:Class ;
  rdfs:subClassOf owl:Thing ;
  owl:disjointWith owl:Nothing ;
  owl:equivalentClass <http://www.example.com/seanjames/ontologies/animals#Papilionoidea> ;
  owl:sameAs <http://www.example.com/seanjames/ontologies/animals#Papilionoidea> .
```

Appendix E –Screenshots

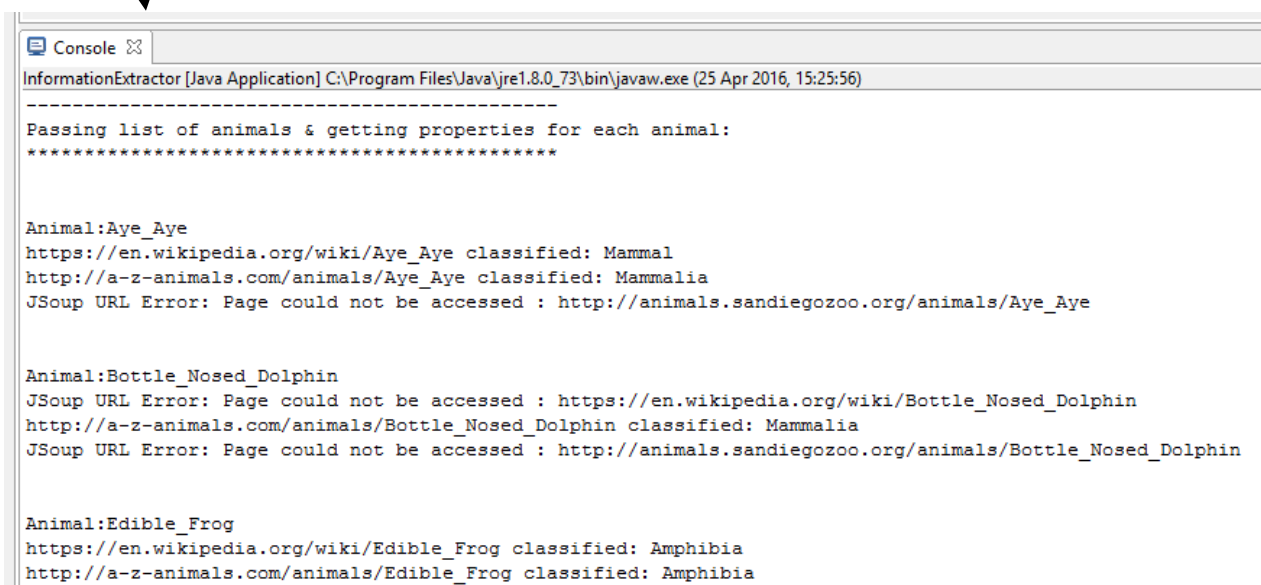
Screenshot E-1: Eclipse IDE

Eclipse was a very useful programmatic tool and offered functionality to test code through “console” output, correct and indicate errors and the ability to modify code in a project structure.



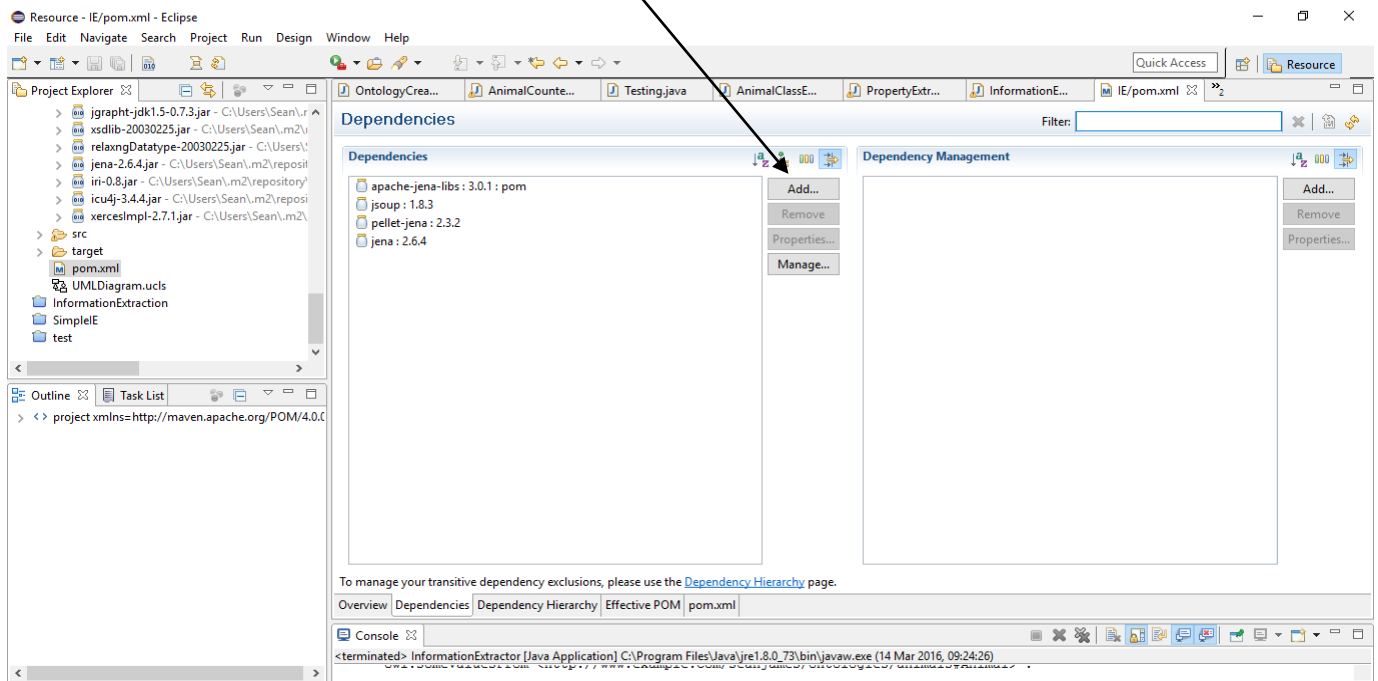
Screenshot E-2: Eclipse Console and system output

The “console” output feature is very similar to a CMD style interface.



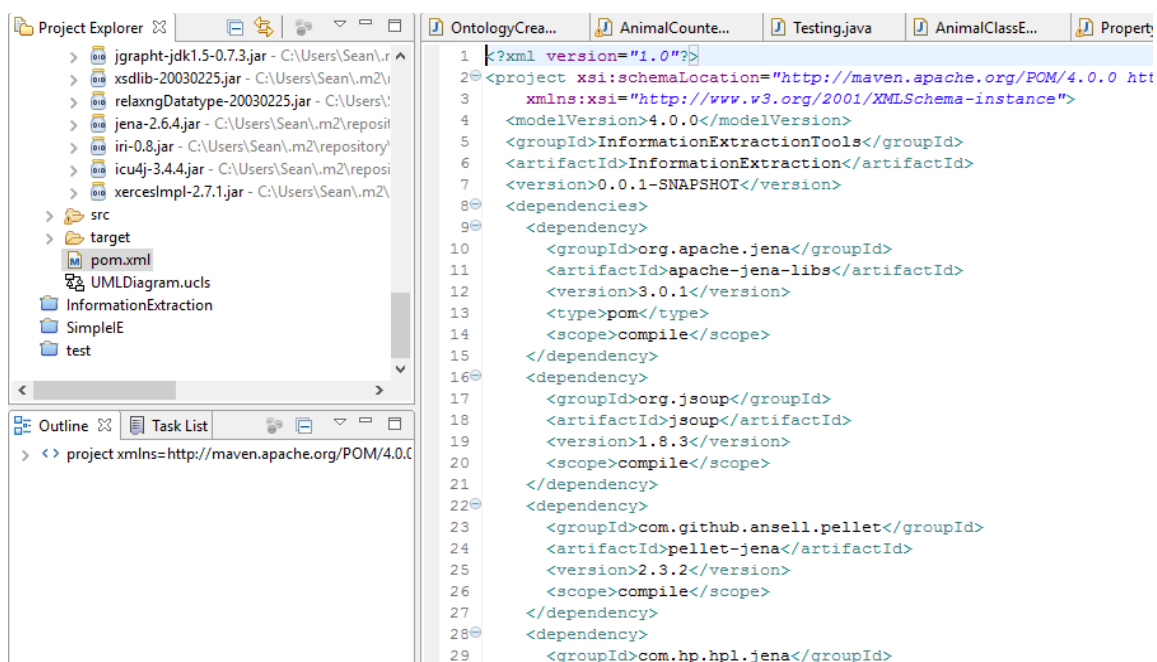
Screenshot E-3: Adding dependencies using Maven

Dependencies can be easily added using the “Add” button, making it straightforward to add libraries (as .JAR files) and maintain dependencies for the solution.



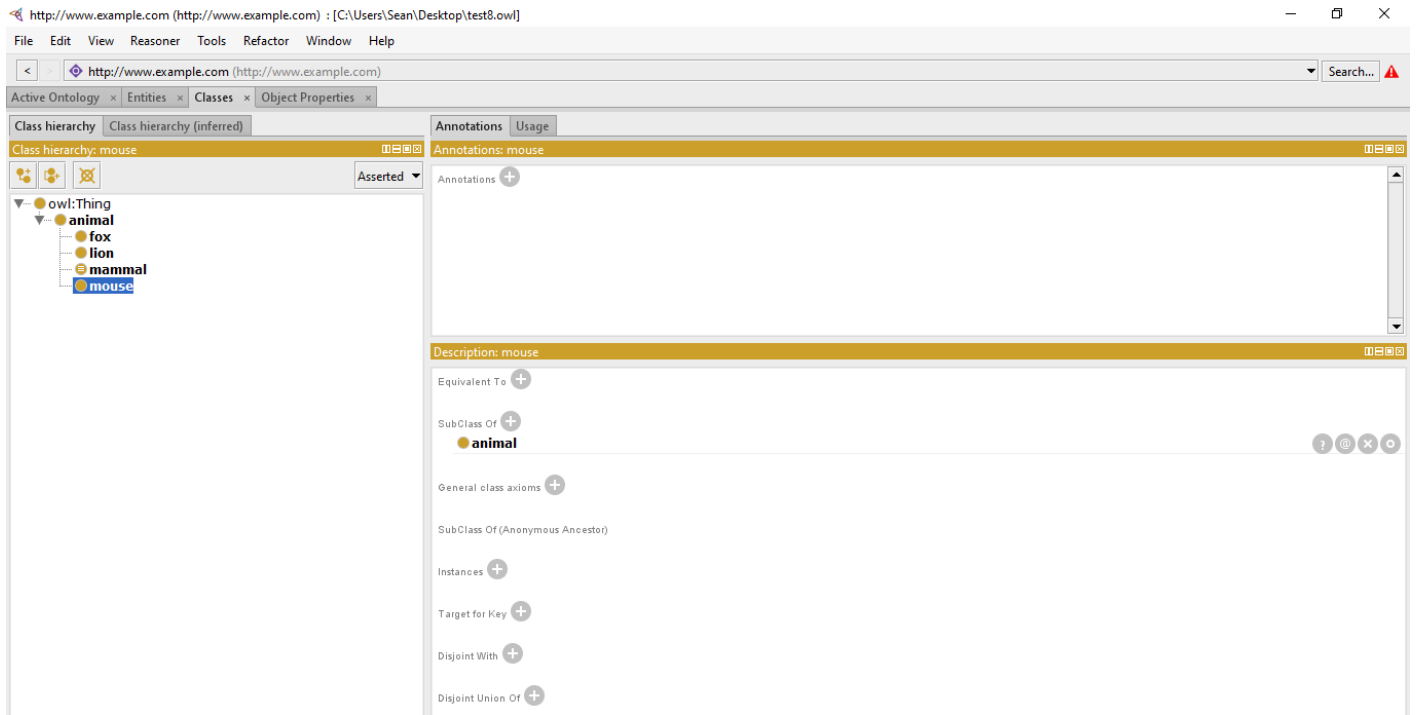
Screenshot E-4: Corresponding Maven POM.XML file

The POM.XML file specifies each of the dependencies and with this file anyone can download the required libraries (such as JENA and Jsoup) to use my solution.



Screenshot E-5: Protégé

Protégé was a very useful tool for modelling and testing ontological models; it allows the addition and creation of properties in a basic GUI manner.



Appendix F – Animal Website Selection Criteria

The following factors were taken into account in selecting websites to be used for this prototype:

Positive Factors

- The site contain animal information suitable for animal information extraction and features suitable patterns for information identification
- The site is available with little or no visible downtime for maintenance
- Contains words to allow property extraction to occur
- Provided Latin names for animals
- Provided information about an animal's prey
- Provided information about the predators of a given animal
- Searching for animal can be facilitated by adding the animal's name to the URL
- The existence of an A through Z list of animals
- Contains an extensive encyclopaedia of animals representing used animal classes e.g. Fish, Mammals, Reptiles, Birds etc
- The website features a search facility to assist with testing
- The website features a useful set of animal facts and figures

Negative Factors


- The site is too specific e.g. rather than Frog uses "Golden Poison Dart Frog"
- The site's HTML is too unstructured to parse effectively
- Excessive unstructured text
- Site contains more exotic animals e.g. Golden Lion Tamarin
- No clear patterns in the text

Website F-1- A-Z Animals Site

The site which offered the most functionality in terms of Information Extraction. The key advantage and difference being the presence of “predators” and “prey” of a given animal. The site offered a substantial amount of useful text and provided a lot of fact type information which could be additionally extracted in future iterations.


Animals >> **Lion** Search

[Add to Phobia Filter](#) [Contribute](#) [Print](#) [Listen](#)



Lion Classification and Evolution

The Lion is one of the largest, strongest and powerful felines in the world second only in size to the Siberian Tiger. They are the largest **cats** on the African continent and are unique among felines in a number of ways but the biggest difference between Lions and other **cats** is that they are incredibly sociable **animals** that live together in family groups known as prides. Lions are also part of the big **cat** family meaning that both males and females are able to roar. Despite having once roamed across much of **Africa** and even parts of **Europe** and **Asia**, the world's remaining Lion population now resides in sub-Saharan **Africa**. However, with Lion numbers thought to have dropped by 30% over the past 20 years these enormous **predators** are sadly becoming more and more vulnerable in their decreasing natural environment.



Lion Facts

Kingdom: Animalia
Phylum: Chordata
Class: Mammalia
Order: Carnivora
Family: Felidae
Genus: Panthera
Scientific Name: Panthera leo
Common Name: Lion

Lion Anatomy and Appearance

Historically, Lions would have been found throughout much of **Africa** and even in parts of **Europe** and **Asia** as well. Today however, they have been pushed into more isolated pockets of their once vast natural range with the remaining African Lion population now only found in countries in sub-Saharan **Africa**. There is also still a small population of Asiatic Lions found inhabiting a remote part of the Gir Forest in India where there are an estimated 300 individuals remaining. Despite their dwindling numbers, Lions are actually incredibly adaptable **animals** that can and will inhabit very dry climates as they get most of the moisture they need from their food. They prefer areas of open woodland, scrub and long **grasslands** where there is not only plenty over cover but also a wide variety of **prey**. They are only not found in areas of **rainforest** or far into **deserts**.

Skin Type: Fur
Size (L): 1.4m – 2.5m (4.7ft – 8.2ft)
Weight: 120kg – 249kg (264lbs – 550lbs)
Top Speed: 56kph (35mph)

Diet: Carnivore
Prey: Antelope, Warthog, Zebra
Predators: Human

Useful Food Chain Information: Predators and Prey

The next two sites were only used for extracting the classification and finding properties. Predator and prey information was not available and so they were not used for “Food Chain” processes.

Website F-2- sandiegozoo.org site

A useful site with plenty of possibilities for extension, such as using “tweet” information to add further details to a given model. The detailed text present on this time was good for property classification and although the class extraction required some standardisation “Mammalia (Mammals)” to “Mammal” it was very effective overall.

Website F-3- Wikipedia

It was useful for initial testing purposes because it was a familiar and complete resource. It offered “animal classification” information which was almost normalised. The necessary patterns were present to perform effective information extraction making it a good site for testing whether techniques were feasible. It was accepted that it was not the best resource because of its public nature and so any results obtained were closely examined.

References

- *agile software development* (ASD) [Online]. Available at: <http://searchsoftwarequality.techtarget.com/definition/agile-software-development> [Accessed 30th April 2016]
- Anzaroot Q.L, Lin W.P, Li.X, Ji.H. *Joint Inference for Cross-document Information Extraction* [Online]. Available at: http://nlp.cs.rpi.edu/paper/reasoning_cikm.pdf [Accessed 25th February 2016]
- Apache Jena [Online]. Available at: <https://jena.apache.org> [Accessed 10th February 2016]
- Apache Maven [Online]. Available at: <https://maven.apache.org> [Accessed 10th February 2016]
- Eric Schmidt Quote: *Every 2 days we create as much information as we did up to 2003.* [Online]. Available at: <http://www.azquotes.com/quote/921446> [Accessed 20th February 2016]
- Fonou-Dombeu J and Huisman M. 2011. *Combining Ontology Development Methodologies and Semantic Web Platforms for E-government Domain Ontology Development* [Online]. Available at: <http://arxiv.org/ftp/arxiv/papers/1104/1104.4966.pdf> [Accessed 13th March 2016]
- Geoffrey Moore Quote: *Without big data, you are blind and deaf and in the middle of a freeway* [Online]. Available at: <http://www.azquotes.com/quote/641515> [Accessed 20th February 2016]
- *GitHub* [Online]. Available at: <https://github.com/>
- Gravelle. R. *Web Page Scraping With Jsoup* [Online]. Available at: <http://www.htmlgoodies.com/html5/other/web-page-scraping-with-jsoup.html> [Accessed 11th March 2016]
- Janevski A. 2000. *UniversityIE: Information Extraction From University Web Pages* [Online]. Available at: http://uknowledge.uky.edu/cgi/viewcontent.cgi?article=1219&context=gradschool_theses [Accessed 2nd February 2016]
- Jani.K and Chavda.V. 2014. *A Study on Semantic Web Framework: JENA and Protégé* [Online]. Available at: https://www.worldwidejournals.com/ijar/file.php?val=January_2014_1388583791_ee78c_43.pdf [Accessed 11th March 2016]

- *jsoup: Java HTML Parser* [Online]. Available at: <http://jsoup.org/> [Accessed 10th February 2016]
- Kaiser K, Miksch S. *Information Engineering Group: Information Extraction*. [Online] Available at: http://publik.tuwien.ac.at/files/pub-inf_2999.pdf [Accessed 26th February].
- Lakeworks: *The Scrum Process*[Online].Available at: https://en.wikipedia.org/wiki/Scrum_%28software_development%29#/media/File:Scrum_process.svg
- *LevenshteinDistance.java* [Online]. Available at: <https://commons.apache.org/sandbox/commons-text/jacoco/org.apache.commons.text.similarity/LevenshteinDistance.java.html> [Accessed 21st March 2016]
- McGuinness D.L, Harmelen F. *W3C: OWL Web Ontology Language Overview* [Online]. Available at: <http://goo.gl/9VrWxK>.
- *Protégé*. [Online]. Available at: <http://protege.stanford.edu/>
- *Protégé: Protege-OWL Reasoning API*. [Online] Available at: <http://protegewiki.stanford.edu/wiki/ProtegeReasonerAPI>
- Ratwani R.M, Trafton J.G, Boehm-Davis D.A. *From Specific Information Extraction to Inferences: A Hierarchical Framework of Graph Comprehension* [Online]. Available at: http://www.nrl.navy.mil/itd/aic/sites/www.nrl.navy.mil.itd.aic/files/pdfs/ratwani_0.pdf [Accessed 25th February 2016]
- Sas. *Big Data* [Online]. Available at:http://www.sas.com/en_us/insights/big-data/what-is-big-data.html
- seth, *SETH is a software effort to deeply integrate Python with Web Ontology Language* [Online]. Available at: <http://seth-scripting.sourceforge.net/> [Accessed 20th March 2016]
- Sigletos G,Paliouras G, Spyropoulos C, Hatzopoulos M. *Mining Web sites using wrapper induction, named entities and post-processing* [Online]. Available at : <https://km.aifb.kit.edu/ws/ewmf03/papers/Sigletos.pdf> [Accessed 11th March 2016]
- Stephen Hawking, 2014 *Notable Quotes*. [Online] Available at: http://www.notable-quotes.com/h/hawking_stephen_ii.html [Accessed 1st May 2016]
- TechTarget. *Natural language Processing* [Online] Available at: <http://searchcontentmanagement.techtarget.com/definition/natural-language-processing-NLP>

- The University of Sheffield. *GATE Information Extraction* [Online]. Available at: <https://gate.ac.uk/ie/> [Accessed 25th February 2016]
- W3C: *Resource Description Framework (RDF)* [Online]. Available at: <https://www.w3.org/RDF/>