Human Computation for Ontology Refinement

Suaad Salem Yousuf Bussanad Alshamsi

Master of Science in Knowledge and Data Management
School of Informatics

The British University in Dubai

June, 2008

Abstract

Ontology plays an important role in enabling the Semantic Web. Ontologies are used to represent a shared and common understanding of a domain of interest. The main bottleneck is knowledge acquisition in a dynamic environment like the Web. There are several projects that attempts to collect commonsense knowledge, but none of them has explored how to present and use this knowledge in real life. This project explores the possibility of using human computation concepts to devise an online game that enables people to contribute to the process of generating Semantic Web ontologies. The project presents the *Ontology Refinement System* which generates ontologies from the knowledge acquired by the online game. The system is divided into three main components. The first component is the *Online Game* which is used as a tool to collect commonsense knowledge from a large community of people. The second component is the *Ontology Builder* which represents an algorithm used to illustrate how to process and transfer the knowledge collected into a set of statements. This component is the prime contribution of the project. The third component in the system is the *Ontology Representation Generator* which takes the set of statements produced by the second component and generates OWL ontology files using Jena API. An experiment has been conducted to evaluate the efficiency of the *Ontology Refinement System.* Three different ontologies were produced by the system from the knowledge collected during the experiment. A survey about the ontologies produced by the experiment has been sent randomly to Internet users to measure the accuracy of these ontologies. The survey results in a good agreement from the Internet users on the ontologies produced. The ontologies that are produced by the *Ontology Refinement System* are particularly relevant to domains in which ontologies change over time such as electronic commerce. Ontologies can be used to maintain a constantly evolving and improving product catalogue. A good product catalogue can make it easier for people to navigate the available products to find information.

Acknowledgements

I would like to express my gratitude to all those who made the completion of this thesis possible.

I am grateful to my supervisors Dr. Iyad Rahwan and Dr. Shereif Abdullah at BUiD for their guideness, support and inspiration during the research and writing of the thesis.

I am indebted to all the students in Dr. Iyad Rahwan and Dr. Shereif Abdullah classes for being the volunteers of the experiment which enabled me to evaluate my work.

I wish to thank my colleague Franklin Antony from the IT department at DATEL Systems and Software for his continuous support and encouragement, and for being a good listener throughout the whole thesis.

I would like to thank my friend Khadijah Al Bahri for lending me all the Mathematical books that I needed during the thesis and for her continuous support and encouragement.

I would like to express my deep and sincere gratitude to my parents, brothers, and sisters whose love, support and care enabled me to complete this work. A very special thanks to my mother Moza Al Nuaimi for every little thing she did for me since my childhood to become the person who I am today.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Suaad Salem Yousuf Bussanad Alshamsi*)

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Overview

1.1 Introduction

In recent years, the vision of the Web has evolved from today's Web on which content is suitable for human consumption only, to a Semantic Web on which content is suitable for human consumption and machines processing [8]. In the Semantic Web, the meaning of the information that the Web presents will play a significant role. The search engines will not be keyword–based, but rather context–based. Therefore, the result of the query will be more accurate and relevant to the query. Moreover, the query result can be processed by machines as well as human beings.

The Semantic Web will not be an independent new Web but it will gradually evolve out of the existing Web. The main limitation of the current Web is that the structure of the information is weak. Therefore, the first step toward a Semantic Web is to organise the information in conceptual spaces according to its meaning.

Ontology is one of the Semantic Web technologies that can be used to reorganise the information on the Web based on its meaning. It is a fundamental backbone technology for the Semantic Web [12]. It is used to represent a shared and common understanding of a domain of interest. Ontology is useful for (i) the organisation and navigation of the Web sites, (ii) improving the accuracy of the Web search, and (iii) generalising and specialising the search query if a search fails.

The development process of ontologies is a collaborative process that requires a large community of people to agree on the domain knowledge being represented. This representation should be communicated between people and application systems [10, 29, 12]. In addition, the development process of ontologies is a dynamic process simply because knowledge is not static. Concepts in a domain continuously evolve and ontologies must always adopt these changes [21].

1.2 Statement of the Problem

Currently, the number of useful ontologies that are published on the Internet is limited and the majority of them are outdated [12, 21]. The reason for this is that the current approach for developing ontologies is engineering–oriented meaning that a small group of people build the ontology and then release it to a wider community of users. Therefore, the user community does not have control over the evolution of the ontology. They can not add new concepts or modify existing ones. The user community always depend on the small group of people who have developed the ontology. Moreover, understanding of the intended use of the concepts is hard for the user community since concepts are expressed in a formal language [5, 21].

There have been a number of attempts to develop ontologies through the collaborative ontology engineering approach [21, 27, 30]. However, it is time consuming and requires dedicated users who have the incentive to participate in the development process of ontologies. These users are either paid experts or unpaid volunteers.

1.3 Research Questions

This thesis attempts to answer the following questions:

- How can a large community of people get involved in the process of developing up–to–date ontologies in a short period of time without any cost?
- How can the content of the ontologies produced evolve dynamically without the need to always feed the ontologies with entries?
- How can the commonsense knowledge collected by various projects be processed and presented in a way that can be useful in other domains?
- How can the ontologies produced be filtered from noise? And how can they represent knowledge without redundant and implicit knowledge?

1.4 Contribution

The contributions of this research project are (1) to define an interactive and collaborative means to collect commonsense knowledge, and (2) to propose an algorithm that can process and transfer the knowledge collected to a formal ontology representation. This section outlines the answers to the research questions.

- How can a large community of people get involved in the process of developing up–to–date ontologies in a short period of time without any cost?

Human computation [1] can be easily used as a collaborative ontology development environment, reducing the lack of user's participation in the creation and maintenance of ontologies, and speeding up the process of developing updated ontologies.

This project explores the use of human computation concepts to devise an online game that enables a large community of people to get involved in the process of generating Semantic Web ontologies. People will be playing the game in their leisure time. The more the people are playing the game, the more updated ontologies are built. The *Game Component* is the first component in the project. This component can be replaced at any time if there is any other interactive and enjoyable means to collect commonsense giving the project more flexibility and customisability.

- How can the content of the ontologies produced evolve dynamically without the need to always feed the ontologies with entries?

The initial content of the game i.e. ontologies intended to be produced is predefined. However, once the players agree on concepts the content of the game will change dynamically to include the players' concepts. This change is based on a set of rules and configured parameters that are defined in the game. The game does not depend on any other resource to continuously feed the game like the OntoGame [13]. Therefore, to change the language used to present the concepts in the game only the initial content should be changed to the perspective language.

- How can the commonsense knowledge collected by various projects be processed and presented in a way that can be useful in other domains?

The *Game Component* is only the first step toward facilitating the development of the updated ontologies. It is just a means for collecting commonsense knowledge from a large community of people. This knowledge needs to be processed and transformed to a proper form of ontologies. In order to do this a second component has been developed in this thesis which is the *Ontology Builder Component.* This component represents an algorithm used to process the knowledge collected from the game, and produce a set of statements. This component is also independent and can be modified or replaced at any time.

The third component in this thesis is the *Ontology Representation Generator Component.* It is an application that takes the set of statements produced by the second component and generates an ontology file in OWL format. This component can produce ontologies in any other format as well such as the RDF format. It is also flexible and can be replaced based on the need of how the set of statements should be represented. For example, the set of statements can be used to feed an agent knowledge base.

- How can the ontologies produced be filtered from noise? And how can they represent knowledge without redundant and implicit knowledge?

The *Ontology Builder Component* defines an algorithm that uses a set of thresholds that are considered while processing and transferring the knowledge collected to filter the noise, and remove redundant knowledge.

1.5 Scope

The ontologies that are produced by the game i.e. algorithm, are meant to be simple ontologies that represent commonsense knowledge. The ontologies produced represent only the fundamental and core aspects in an ontology which are concepts and the hierarchical relationship between concepts. In other words, they only represent the sub–class and super–class relationships between concepts. This is because it is the most important aspect for reasoning. Any extension of the game can adopt other kinds of relationships easily.

In addition to the concepts, ontologies can contain other information such as properties and data constrains. This information is out of the project scope and any extension of the game should cover them easily.

Moreover, ontologies can be represented in different formal formats. OWL–DL is the format used to represent the ontologies produced. Changing the format is feasible through modifying the third component. Furthermore, the ontologies are presented in English. As mentioned earlier to change the language, only the initial data need to be changed.

1.6 Organisation of the Thesis

The rest of the thesis is organised as follows. The background chapter reviews the concepts of ontology and human computation. The literature survey chapter explores the work of others which is related to the project. The specification chapter describes the proposed methodologies for collecting commonsense knowledge and building the ontologies. The implementation chapter gives the details of the technical implementation of the methodologies used in this research project. The experimental design chapter describes the design of the experiment used to evaluate the game and the algorithm used to build the ontologies. The evaluation chapter presents and analysis the results of the experiment. It discusses the design and result of the survey that has been conducted to evaluate the ontologies produced from the knowledge gathered during the experiment. Finally, the last chapter of this thesis is a summary of future work that needs to be undertaken and the conclusion.

Chapter 2

Background

This chapter provides basic knowledge about the fields of ontology and human–based computation on which the project is based. It describes the ontology, illustrates the process of building the ontology, and explores the main languages of writing an ontology. In addition, it explores the field of human–based computation specially the "Games with a Purpose" methodology.

2.1 Ontology

Ontology is a fundamental backbone technology of the Semantic Web. It enables knowledge to be organised in a structured form. Moreover, it facilitates knowledge sharing and reuse. Ontology aims to organise and represent understandable domain knowledge in a way that can be communicated between human beings and machines.

This section explains briefly the meaning of ontology and the main languages used to represent the ontology formally.

2.1.1 What is an Ontology?

The term ontology originated in the subfield of Philosophy that is specialised in studying the nature of existence. In recent years, the term ontology has been also used in the field of Computer Science. In this field, the term ontology can be defined as a formal, explicit, specification of a shard conceptualisation [29]. Conceptualisation refers to the fact that the ontology is an abstract model that represents a specific domain in the world. The model contains all the relevant concepts of the domain. Explicit refers to the fact that the type of concepts in the model and the restrictions of their use are explicitly stated. Formal refers to the fact that the model is represented in a way that is understandable for human being and machines. Shared refers to the fact that the model is not meant to be owned by individuals but rather it should be defined and agreed upon by a group of people. In practice, ontologies are represented by a special kind of graph.

Ontologies consist of a list of finite concepts and relationships between these concepts. Concepts represent the most important aspects of a domain while relationships represent the type of connection between these concepts. For instance, in the domain of automobile, concepts would include car, truck, engine, and coupe while relationships would include is–a–subclass–of and is–part–of like coupe is–a–subclass–of car, and engine is–part–of a truck. The project deals only with the core relationship in an ontology which is the hierarchical relationship i.e. is–a–subclass–of and is–a–super–class–of. This type of relationship is used by reasoning engines to infer further statements about the domain. Any other relationship type can be added similarly.

Furthermore, ontology can include other information such as (i) properties e.g. X drives Y, (ii) value restrictions e.g. only one driver can drive a car, (iii) disjoint statements e.g. driver and passengers are disjoint, and (iv) specification of logical relationships between objects e.g. each car must at least have two doors. This information is out of the project scope.

Figure 2.1 represents the main components of an ontology and how these components interact with each other.

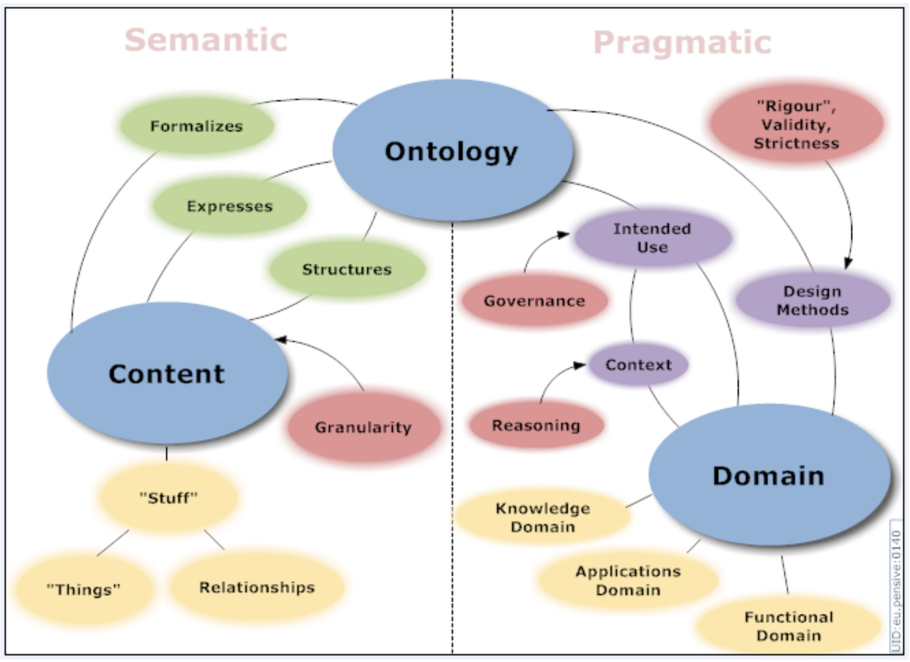


Figure 2.1: Ontology Dimensions Map [25]

2.1.2 The Process of Building an Ontology

There is not a single method of describing a domain of interest as it depends on the understanding and scope of the domain. The process of developing an ontology is iterative. Following is a step–by–step description of how an ontology can be built as stated in [23].

- Determine the scope and domain of the ontology
- Consider reusing existing ontologies

- Enumerate important terms in the ontology
- Define the classes and the class hierarchy
- Define the facets of the properties i.e. the value type, allowed values, cardinality, and other features of the values the properties can take
- Define the properties of classes
- Create instances i.e. individuals of the class

In addition to the above steps Uschold and King [22] have stated in their methodology that after building an ontology, it should be represented formally using one of the ontology's languages. Furthermore, Uschold and King have mentioned that the ontology produced should be evaluated to verify whether it met its requirements.

The evaluation of an ontology can be accomplished in different approaches. One approach is to measure the similarity between the ontology produced and other existing ontologies in terms of their lexical and conceptual content as described in [3]. Brank and Grobelnik [11] compared various techniques to evaluate an ontology such as comparing the ontology to a "golden standard", using the ontology in an application and evaluating the results, and comparing the ontology with a source of data about the domain to be covered by the ontology. Moreover, PROMPT which is a suite that can be integrated with Protégé[1], can be used to measure the alignment between two ontologies [24]. This tool can be used to evaluate the ontology produced as well. In addition, a survey rating the accuracy and efficiency of the ontology produced can be used as a tool to evaluate the ontology.

Finally, Uschold and King [22] have mentioned that the ontology produced should be documented explaining all the definitions and assumptions for the ease of knowledge sharing.

2.1.3 What are the Ontology Languages?

There are many languages that can be used to write an ontology. The most important languages are listed below [8]:

- XML provides a standard for writing a structured document but it does not provide any standard for describing the meaning of the document.
- XML Schema is a language for defining the structure of the XML documents.
- RDF is a graph–based data model for describing objects i.e. resources. It provides simple semantic for the data model. It can be represented in XML syntax.
- RDF Schema is a vocabulary description language used to describe the properties and classes of the RDF resources. It is limited to a subclass hierarchy and property hierarchy with the domain and range definitions of these properties.

[1]Protégé is a free, open source ontology editor and knowledge-base framework.

- OWL is a richer vocabulary description language. It is used to describe properties and classes from different perspectives such as relations between classes, cardinality, and equality.

OWL language is used in this project to document the ontologies produced. OWL language has three extensions that can be used based on the different requirements. OWL Full is the entire OWL language. It is fully upward–compatible with RDF, both syntactically and semantically. It is very powerful to the extend that it does not have a complete or efficient reasoning support. OWL–DL (Description Logic) is a restricted version of OWL Full. It does not have full compatibility with RDF but it has efficient reasoning support. OWL–Lite is a restricted version of OWL–DL. It excludes enumerated classes, disjointness statements, and arbitrary cardinality. OWL–DL is the extension that is used in this project.

2.2 Human–Based Computation

Human–based computation simply means humans taking part in solving computer problems. In traditional computation, humans use the computer to solve problems. Humans describe the problems and computers provide solutions. In human–based computation, the roles are often reversed. The computer asks a person or large number of people to solve a problem. The computer then collects, processes, and integrates these solutions [1].

Human–based computation initiated because of the fact that computers are lacking the basic conceptual intelligence and perceptual capabilities of the human beings. For humans to take part of the computational process, they need to have the incentive to do so. There are different approaches to motivate people to participate in the human computation. Wiki is a method of human computation that can be used by people who have the incentive to communicate and share knowledge. Interactive online guessing games are another method of human computation. These games extract knowledge from the players who are playing the game. The knowledge exists in the answers the players are entering. These games have been refereed as “Games with a Purpose” by Luis von Ahn [16]. This method can be used by people who have the competitive spirit of a game.

The “Games with a Purpose” methodology is used in this project to solve the problem of generating ontologies. The next section describes the “Games with a Purpose” in more details. It presents examples of some games and shows statistics of the games’ success.

2.2.1 Games with a Purpose

“Games with a Purpose” is an online game that uses the human power to solve open world problems that can not be solved by computers alone. The game should solve a clearly defined problem. The design of the game is similar to the design of an algorithm. The input and the output of the game should be stated clearly and precisely. The design of the game should take into account that people are not playing the game to solve world problems but to entertain themselves. Therefore the game should be enjoyable.

Moreover, it must be ensured that the output of the game is correct. Luis, has followed various strategies in order to ensure the accuracy and efficiency of the output. One of the strategies is random pairing. The players are randomly assigned to play with an anonymous pair. Therefore, the possibility of cheating is eliminated. Another strategy is the single player mode. In this mode, the player is playing against the computer i.e. a previous game. This helps in validating the knowledge collected in the previous games. In addition, the time the players takes to find a solution is used as an indication of the quality of their answers [18].

"Games with a Purpose" has many potential applications in areas such as artificial intelligence, computer vision, security, ecommerce, and social networking. Luis developed four online games which are the ESP, Peekaboom, Verbosity, and Phetch. The following section explores these games in more details.

Examples of Games with a Purpose

The **ESP** game is an enjoyable online game that aims to label images on the Web. These labels can be used to improve the accuracy of image search, improve site accessibility, and help users in filtering improper images [17].

The **Peekaboom** game is another interactive online game that aims to locate objects in images. The data that is collected from the game can be used to train computer vision algorithms. These algorithms need massive data to be trained and tested which is currently not available [19].

The **Verbosity** game is an entertaining online game that intends to collect commonsense knowledge. The knowledge collected in this game can be used mainly in the field of artificial intelligence [18].

The **Phetch** game is another interactive online game. The purpose of the game is to collect statements that describe images on the Web. These statements are indented to provide proper captions for the images on the Web. These captions facilitate the accessibility of images on the Web to visually impaired people [20].

Notice that the purpose of the games described above is simple and precise. However, the contribution of the data collected is valuable in terms of solving open world problems.

The potential data that can be collected from these games is huge. As stated in [16] the **ESP** game alone collected more than 10 million image labels in few months of its initial deployment. Luis believes that if the game was deployed in popular Web sites like MSN or Yahoo, the game would have been able to label all the images on the Web in weeks [16]. Moreover, the **Peekaboom** collected 2 million pieces of data since it has been released [16].

Chapter 3

Literature Survey

There are different related works to this project. These works fall into one of the following categories: (i) folksonomy or collaborative tagging, (ii) collaborative ontology engineering, (iii) commonsense gathering, (iv) human computation to collect commonsense, and (v) human computation to build ontologies. This chapter briefly introduces each field, presents the limitations of each field, and finally illustrates how this project can contribute in overcoming these limitations.

3.1 Folksonomy

Folksonomy or collaborative tagging is a means of classifying content on the Web such as Web pages, photographs, Web links using unrestricted tags i.e. labels. These tags are generated through the collaborative efforts of the Internet community [28]. The objective of folksonomy is to facilitate content retrieval through sharing and organising knowledge. Folksonomy uses free vocabulary tags in order to describe content. Examples of tagging systems are **del.icio.us**, **flicker**, and **bibsonomy**.

del.icio.us[1] is a social bookmarking Web site which enables Internet users to store and share bookmarks online. Internet users can add and retrieve their bookmarks from any computer. Moreover, they can use tags to organise and remember their bookmarks. Internet users can view other people bookmarks and share their bookmarks with them.

Flicker[2] is a photo sharing and annotating Web site. It allows Internet users to add and share photos on the Web. Internet users can use tags to organise the published photos. Tagging facilitate browsing and searching the various photos.

Bibsonomy[3] is similar to del.icio.us. It is a social bookmarking Web site specialised in classifying scholarly publications. It allows Internet users to store and share bookmarks and publication entries. In addition, it enables the Internet users to use tags to organise the publications and facilitate their retrieval.

[1]http://del.icio.us/
[2]http://www.flickr.com/
[3]http://www.bibsonomy.org/

In spite of its popularity, folksonomy has several limitations as described in [4, 5, 15]. The first limitation is that folksonomy does not force any rule when introducing new tags. Internet users are free to add any tag they want. For example, users can add “NY”, “N.Y”, “New York” to describe New York. This leads to inconsistency and redundancy of tags. Moreover, it makes information retrieval hard as the user should be aware of all the possible variations of a word to get all the resources. The second limitation is that the tags are in a raw form. They do not have any meaning associated with them. As a result the tags are ambiguous and context dependent. For instance, the tag “Spring” may refer to the season of growth, or a point at which water issues forth, or the Java Framework. The last limitation is that folksonomy consists of flat tags that have no relationships defined among them. Therefore, an object can be described with general and special tags. For example BMW can be tagged as BMW and car. Car might be too general to some users while BMW might be too specific for others. In addition reasoning is impossible.

As presented in [4] and [15], in order to overcome the limitations of folksonomy, ontology should be added as a layer on top of it.

3.2 Collaborative Ontology Engineering

OntoWiki uses a collaborative ontology engineering approach to enable Web users to participate in the ontology development process and does not limit the development to a small community [21]. It does not require the Web users to have any expertise in ontology engineering. It only requires them to have basic Web editing skills.

OntoWiki is suitable for complex Ontologies. However, it is time consuming. Users should be willing to spend time to agree on a concept definition through non–structured discussion [27, 12]. It does not show any evaluation or monitoring criteria for the content being added to the wiki. Moreover, it does not illustrate the methodology of how the ontology can be formally presented and used and how the various discussion pages and versions can be integrated.

3.3 Commonsense Gathering

Commonsense is true statements about the world. It is an important aspect in Artificial Intelligent (AI) systems, especially in expert systems. There are several projects that are intended to gather commonsense knowledge such as **Open Mind** [26], **Mindpixel** [2] and **CYC** [7]. These projects use the Internet for distributed knowledge gathering, particularly commonsense knowledge. However, as stated in [18] they have not been able to collect a large enough portion of commonsense knowledge. This is because they require dedicated individuals who have the incentive to enter these statements. Those individuals are either paid experts or unpaid volunteers.

3.4 Human Computation to Collect Commonsense

There are two other projects that aim at collecting commonsense knowledge using human computation. These projects are Verbosity and Common Consensus.

Verbosity is a Web–based game that has been developed by Luis von Ahn and his colleagues in order to collect commonsense facts in an enjoyable environment. People are playing the game for entertainment without knowing the main purpose of the game [18]. This is the secret of the game's success.

Common Consensus is another Web–based game that collects commonsense knowledge about everyday goals [9]. It is based on the idea of Verbosity but it is specialised in collecting goals only. This game has been evaluated by a small number of users. Thus, the success of the game is doubtful in terms of its ability to collect indented knowledge and whether the game is in fact enjoyable.

The aim of both of the games is to motivate people to collect commonsense knowledge in an enjoyable way without their knowledge. However, neither of them demonstrate how this knowledge can be processed and used.

3.5 Human Computation to Build Ontologies

OntoGame [13] is an online multiplayer game aimed at building ontologies in a collaborative environment. The game is inspired by Luis von Ahn's "Games with a Purpose". Players are presented with random Wikipidia articles and asked to first classify the article as class or instance and if they agree on the answer they are asked to find a common abstraction of the Wikipidia article i.e. give a label to the class or instance.

The **OntoGame** demonstrates how the required knowledge of building ontologies can be collected, but it lacks the real transformation of the knowledge to ontologies. Moreover, it does not define how the knowledge collected from a single game or multiple games can be integrated to produce the ontologies. The core relationship in an ontology which is the hierarchical relationship is missing.

In addition, the game requires the players to have some knowledge about aspects of ontologies such as class, individual, and relation. Furthermore, the data that is used to build ontologies are Wikipedia articles. These articles cover complex knowledge and not commonsense. It is the only source to feed the game, therefore no dynamic data is used such as players input. As a result the game can be hard for non–English speakers and teenagers as it requires understanding of the Wikipeida articles. Besides this understanding, it requires considerable reading which is not interesting for many people.

3.6 Proposed Solution

In this proposed project, the *Ontology Refinement System* is developed to construct basic commonsense ontologies that can be used as a basis for more complex ontologies. Ontologies are generated to overcome the limitations of folksonomy.

In contrast to the commonsense gathering and collaborative ontology engineering, the *Ontology Refinement System* uses human computation in order to gather commonsense knowledge and build ontologies. Therefore, commonsense knowledge is gathered explicitly in an enjoyable collaborative environment. The ontologies that are constructed in the *Ontology Refinement System* will evolve as long as people are playing the game. The players do not need to dedicate time to participate in building the ontologies. They can play in their leisure time.

It is believed that the more enjoyable and efficient the game is, the faster ontologies can be constructed. The project combines the enjoyment of the game and collaboration required to build useful ontologies. The collaboration comes from people playing together and agreeing on the same answers.

Unlike the **ontoGame** that always requires feeding the game with articles, the *Ontology Refinement System* adds dynamic content to the game from the players input in a controlled manner. Only the initial data in the game needs to be predefined.

All the related works presented earlier lack an important aspect which is knowledge representation. All these projects focus only on the knowledge acquisition. However, in the *Ontology Refinement System* the knowledge gathered is then processed and transferred implicitly to a set of statements which are then used to generate ontologies.

The key contribution of this project is not only collecting commonsense in enjoyable means, but also processing and illustrating the knowledge in an efficient and effective format.

Chapter 4

Specification

The main goal of the project is to speed up the process of building ontologies with the help of a large community of people. It aims to process and transform the vast amount of the collected commonsense knowledge into proper form of ontologies. As described in chapter 2, there are a number of steps followed during the development of an ontology. The steps that are covered in this thesis are (1) determining the domain and the scope which is predefined by the author, (2) enumerating important terms in the ontology which is achieved by collecting a vast amount of data from a large community of people through the *Game Component*, (3) defining the classes and the class hierarchy which is accomplished through the *Ontology Builder Component* which is an algorithm that builds accurate ontologies based on the data collected from the online game, and (4) representing the ontology formally using one of the ontology's languages which is achieved through the *Ontology Representation Generator Component*.

This chapter explains in details the main specification of the *Ontology Refinement System* and the techniques that have been followed during the development of each component in the system. It provides example or scenario for each component. Moreover, it presents the pseudo code of the algorithm that has been designed and developed for the *Ontology Builder Component*.

4.1 Main System Specification

The specification of the *Ontology Refinement System* is represented in figure 4.1. The system consists of three main components which are the *Game Component*, the *Ontology Builder Component*, and the *Ontology Representation Generator Component*.

The *Game Component* is a tool used to collect the commonsense knowledge from a large community of people and store this knowledge in a database. The *Game Component* is divided into two subcomponents which are the *Game Basics* and the *Game Mechanics*. The *Game Basics* subcomponent is responsible for displaying the game to the players and storing the knowledge collected from the game. The *Game Mechanics* subcomponent is responsible for updating the game database i.e. the question and taboo[1] lists.

[1]Word that the player is not allowed to provide as answer.

The *Ontology Builder Component* is an algorithm that reads the collected commonsense knowledge from the game's database, processes this knowledge, and produces an accurate and minimal set of statements about the collected knowledge. This set of statements is sent to the *Ontology Representation Generator Component.*

The *Ontology Representation Generator Component* is an application that interprets the set of statements sent by the *Ontology Builder Component* and generates an ontology OWL file. The OWL file contains a set of classes and the "is–subclass–of" relationship between these classes.

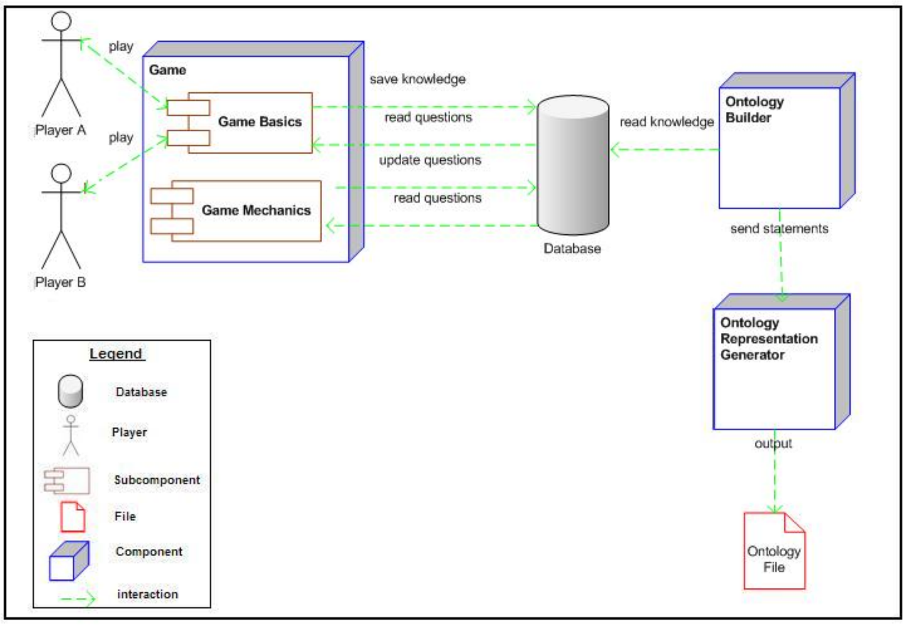


Figure 4.1: Ontology Refinement System's Main Components

As shown in the figure 4.1, the *Ontology Refinement System's* components are loosely coupled with each other. Therefore, any component can be easily replaced by another based on the requirements. The following sections explain each component in more detail.

4.2 Game Component

The technique that has been used to collect a vast amount of data in a short period of time has been inspired by Luis Von Ahn. Luis, the inventor of *"Games with a Purpose"* [16], has demonstrated that online games can be used to solve open world problems. The argument behind this is that people spend billion of hours playing games online. Luis has developed several games that solve different problems. For example, he has developed the ESP game [17] that aims to describe images on the Web. The author believes that building a Web–based game is the best approach to speed up the process of collecting the data required to build ontologies. Moreover, this approach gives the wide community the opportunity to contribute into the process of building ontologies.

The design of the game is similar to the design of an algorithm. The input and the output of the game should be clear. Moreover, it must be ensured that the output of the game is correct and, at the same time it must be ensured that the game is enjoyable. People are playing the game to entertain themselves not to solve problems [16].

Building ontologies is one of the open world problems. This problem is compound and complicated because the ontology contains various features as explained in 2.1. In order to keep the game enjoyable and effective, the problem can be divided into smaller problems. The part that this project aims to solve is building the hierarchical structure of an ontology (i.e. the classes and the subclasses). Any extension of the game can easily consider the non–hierarchical relationships between the classes.

The *Game Component* consists of two subcomponents which are the *Game Basics* and the *Game Mechanics*. The *Game Basics* is in charge of the flow of the game being displayed to the players while the *Game Mechanics* is in charge of the techniques that dynamically update the games database i.e. question and taboo lists. The following sections describe in detail these two subcomponents.

4.2.1 Game Basics

Each player is matched with an anonymous partner i.e. another online player. The two players are presented with concepts which are retrieved from some existing ontology, or previous game. They are then asked to write down all the sub–categories they can think of. The concepts that are presented to the players are from general to specific in order to keep the game attractive and challenging. A player cannot see what his or her partner is typing, but if both write an identical sub–category, they earn points. The players do not need to give the identical answer at the same time. The only restriction is that the answer should be identical for the same question and not when another question is being displayed. The player has the option to pass if the concept presented is hard. The game lasts for two minutes only in order to keep it more interesting. The time is a configurable parameter that can be adjusted before playing the game.

Figure 4.2 presents the flow of the game. A sample scenario of the game is described below:

Player A logs in to the game providing a player ID, Sara. A login request is sent to the server. The server validates the player ID. If the player ID exists i.e. there is another player who is connected to the game server with the same ID, then the server will acknowledge the player with a failure login message which is "Please choose another Player ID and login as the Player ID you selected is already taken". If the player ID does not exist i.e. the Player ID is unique and no other connected players are using the same ID, then the server will acknowledge the player with a successful login message which is "Welcome to the Ontology Game: Sara. You will join a game soon".

If the login is successful, then the server tries to start a new session of the game for Sara. The server searches for a free logged in player i.e. a player who is not engaged in any game yet. If there is no free player, then the server will acknowledge Sara with the following message "Trying to find an opponent", and she will be in awaiting state. As *soon as the server finds a free player, a new session will start and Sara will be playing the game.*

As the game starts, Sara will be presented with a question and a list of associated taboo words. Sara needs to provide answers to the question. The answers should not be identical[2] *to any word in the taboo list. If she provides an answer which is in the taboo list, a message will be displayed to her stating "Answer is a Taboo". Once she provides an answer identical to her partner the question and the taboo list will change and her score will increase. If the question is hard, Sara can send a pass request to her partner. A message stating that "Your Partner is asking to pass" is shown to Sarah's opponent. If her partner agrees to pass, then the question will change. Otherwise, the question will not change and they can keep providing answers to the question. The game will last for two minutes. After that the final score will be displayed for both the players.*

Whenever a session starts, a random set of questions will be retrieved and displayed to the players. This will ensure that the players do not get the same questions each time they play the game. The number of questions that are retrieved is configurable i.e. the value can be adjusted at any time. Whenever a session is over, the information collected from the session is stored in a persistent storage. This information consists of (i) the answers that the players agreed on and their respective questions, (ii) the list of questions they agreed to pass, and (iii) the game score. The answers and the list of questions the players agreed to pass, are used by the *Game Mechanics* in order to update the game's databases. The answers that the players agreed on and their respective questions are processed by the *Ontology Builder Component* to produce ontologies. The score is used by the *Game Basics*. The *Game Basics* displays the highest score whenever a game is over in order to motivate the players.

An interesting feature that has been added to the game is controlling the order in which the questions are displayed to the players. The easy questions are displayed first and then the hard questions. This has been implemented by considering the number of times the players have agreed to pass a question as a measure to categorise the question to be hard or easy. The questions that have been rarely passed are considered to be easy questions and they are displayed first to the players in order to get the players into the game. The questions that have been often passed are considered to be hard questions and they are displayed later to the players.

Examples of the sort of questions that are displayed in the game are:

"____________ is a kind of animal?"
"____________ is a kind of book?"

The phrasing of all the questions is the same. Only the concept e.g. animal, book that is asked about changes. The keyword "is a kind of" has been selected carefully to inspire the player that the relation between the answers he or she is providing and the concept in the question is "a-sub-class-of" relation and not any other kind of relations

[2]Similar is future work. A text processing tool can be used to identify the root of the different variations of a word e.g. noun, verb, adjective, single, plural etc.

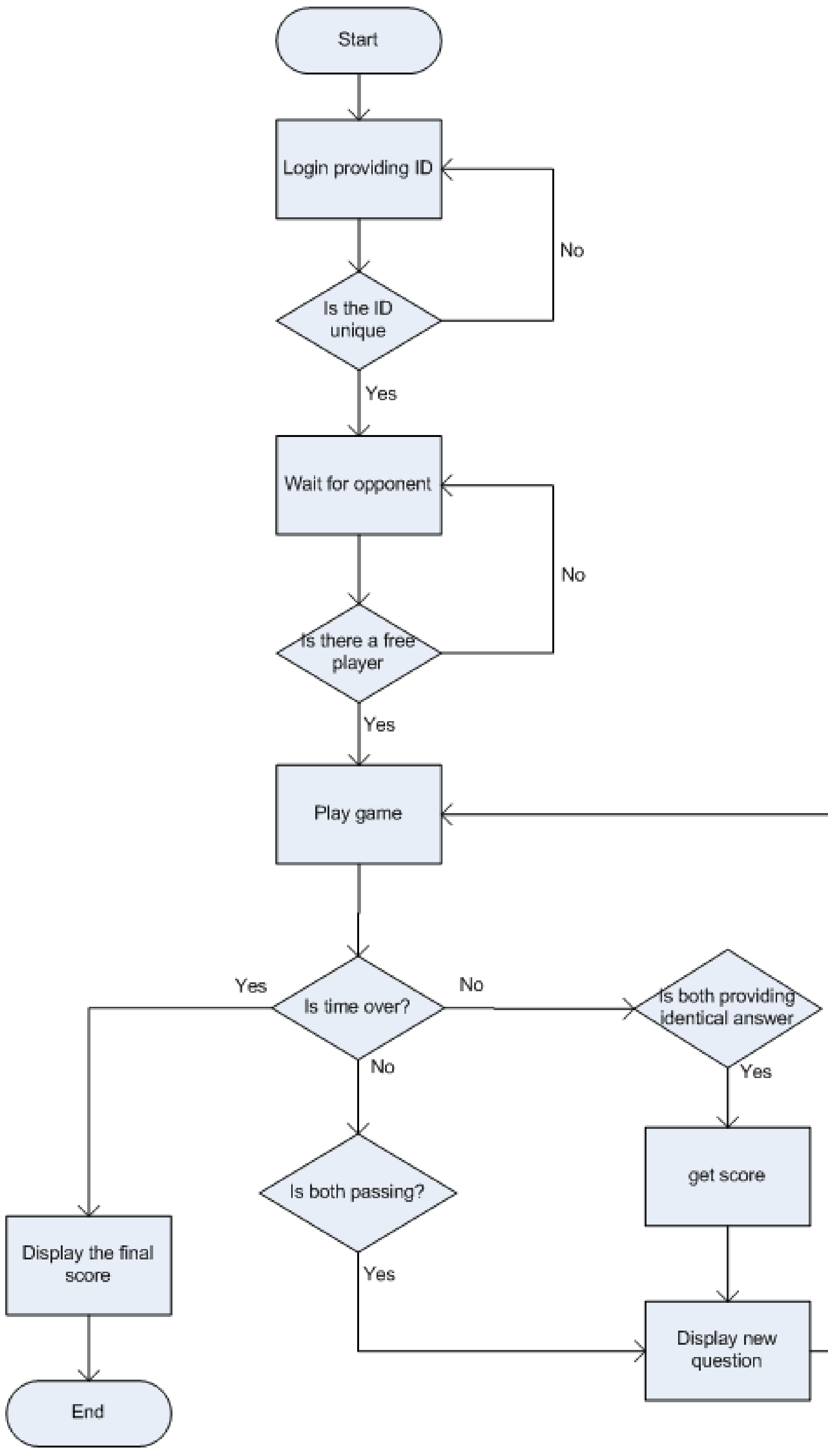


Figure 4.2: Game Flow

like "is-part-of". In addition, the keyword gives the player the impression that he or she needs to provide universal answers to the concept in the question and not concrete answers. The main concern of the project is to build ontologies with the hierarchical structure in it and not to categorise concrete classes or instances.

The answers that the players agreed on might become questions in future rounds of the game based on special criteria which is discussed in the next section. This mechanism assists in building the hierarchical structure of the different ontologies.

Ontologies can overlap and concepts can be shared across ontologies but might be interpreted differently. To keep the player aware of the ontology refereed in each question, the ontology name is displayed as a title along with each question.

The next section explains how the game's database gets updated dynamically through a set of rules and configurable parameters.

4.2.2 Game Mechanics

The game's database should be updated periodically and dynamically based on the players input. New questions can be generated, old questions might be removed from the game, and new elements can be added to the taboo list. A set of rules have been defined in order to update the game's database. The objective of these rules is to update the question and taboo lists displayed to the players during the game so that the players do not get the same set of questions all the time. The set of rules depends on a number of configurable parameters i.e. thresholds. These thresholds can be adjusted at any time. Table 4.1 describes the set of thresholds defined in the *Game Mechanics* subcomponent.

The *Game Mechanics* updates the game's database based on five rules which are:

Rule 1*: Add a question for the answer in which the number of times the players have agreed on that answer is greater than the Question Threshold. This implies that the answer is a valid concept in the ontology hierarchy hence find its sub–classes.*

Rule 2*: If the number of times the players have passed a question is greater than the Pass Threshold then do not display the question any more. This implies that either the question is too hard or the concept can not be specialised any more to sub–classes.*

Rule 3*: If a question has many different answers i.e. the number of answers crossed the Element Threshold, then do not display the question any more. This implies that many sub–classes have been found and the branches have been expanded horizontally, therefore, give the opportunity to other questions to be displayed in order to enable the expansion vertically.*

Rule 4*: If the taboo list of a question is greater than the Taboo List Threshold then do not display the question any more. This implies the same reasoning as rule 3.*

Rule 5*: If the number of times the players have agreed on the same answer to a question is greater than the Taboo Threshold, then add the answer to the taboo list of that* *question. This implies the same reasoning as rule 1. However, in this case it has been added to the taboo list to prevent the players from entering the same answer again in order to give other sub–classes of this concept the opportunity to be discovered.*

Name	Value	Usage
Question Threshold	Represents how many times the players should agree on an answer	Determines when an answer can be added as a question to the question list
Pass Threshold	Represents how many times the players should agree on passing a question	Determines when a question should be removed from the question list
Element Threshold	Represents how many variant answers the players should agree on to a question	Determines when a question should be removed from the question list
Taboo List Threshold	Represents how many variant words there should be in the taboo list of a question	Determines when a question should be removed from the question list
Taboo Threshold	Represents how many times the players should agree on an answer to a question	Determines when an answer should be added as a taboo to the taboo list

Table 4.1: Game Parameters

The thresholds are configurable values that need to be set before updating the game's database. However, the relationship between the thresholds should be taken into account when setting the values of these thresholds. The *Question Threshold* value should be less than the *Taboo Threshold*. The reason for this is to give the answer the opportunity to be a question before adding it to the taboo list and preventing it from being a question ever. This is because once it is added to the taboo list; the players will not be able to provide the same answer as the taboo. Therefore, the value of the answer will never increase to reach the *Question Threshold*.

For example, lets assume that the value of the *Question Threshold* is 10, and the value of the *Taboo Threshold* is 5. If the players agreed 5 times that A is an answer for a question, then according to rule 5 A is added to the taboo list of that question. The players can not provide an answer identical to any word in the taboo list. Thus, the players can not provide A as an answer. As a result, it will never be added to the questions list as the players need to agree 10 times that A is an answer to the question before adding it to the question list.

In addition, the value of the *Question Threshold* affects other parameters that are defined in the second component, *Ontology Builder*. This dependency is explained in the next section.

4.3 Ontology Builder Component

The *Ontology Builder Component* is an algorithm used to build ontologies based on the information stored from the different game sessions by the *Game Basics* subcomponent. The information used is mainly the player's answers. This information is stored as a graph in the database. The graph suffers from three main issues. The first issue is that it contains noise i.e. improper data. The second issue is that it is not minimal. It might contain explicit redundant information which should be removed. For example, the graph might contain the following information:

A is a subclass of B
B is a subclass of C
A is a subclass of C

The third statement is implicitly known since it follows from the first two and from the transitivity of the "sub–class" relation; therefore it should not be present in the graph. The third issue is that the graph might contain cycles as follows:

A is a subclass of B
B is a subclass of C
C is a subclass of A

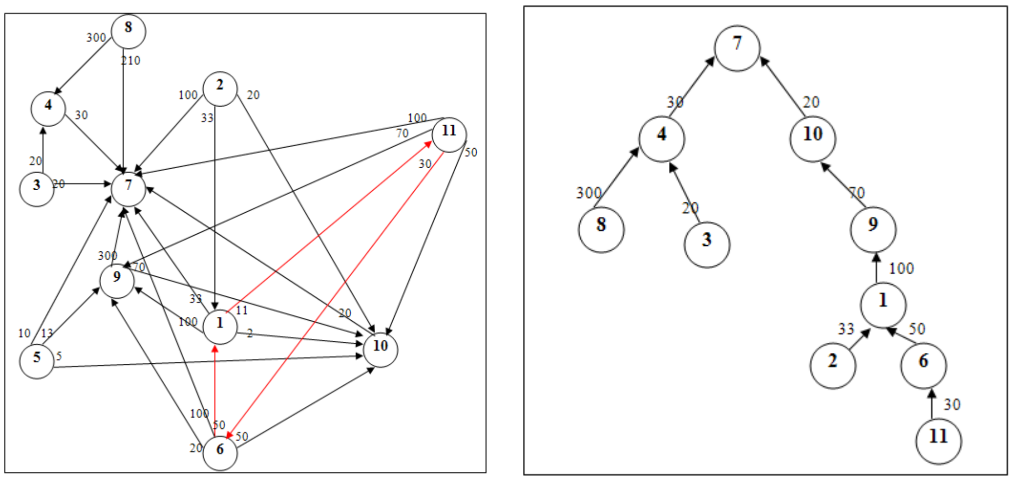
This is not a feature of an ontology. Thus, any cycle in the graph should be traced and removed.

In order to build an ontology, the algorithm takes a weighted directed graph stored in the database and outputs a clean and minimal weighted directed acyclic graph based on certain mechanisms and thresholds. The nodes in the graph represent the answers that the players agreed on while the directed edges represent the "is–subclass–of" relationship between the nodes. The weight in each edge represents the number of times the players agreed that the node A is "a sub–class of" or is "a kind of" the node B by providing answer A to the question that has B as a concept in it. Figure 4.3 shows a sample of the input and the output of the algorithm.

The following section provides a brief introductory to the basic concepts of the graph theory.

4.3.1 Introduction to the Graph Theory

A graph is a set of objects called points, nodes, or vertices connected by links called lines or edges. The set of nodes of a graph **G** is usually denoted by **V** while the set of edges is denoted by **E**. Thus, to indicate that the graph **G** has a set of nodes **V** and a set of edges **E**, it is written as $\mathbf{G = (V,\ E)}$ [14].



(a) Input (b) Output

Figure 4.3: Algorithm Finding

The most important thing to note about an edge is the pair of nodes it connects; hence the edges can be considered as 2-element subsets of **V**. This means that the edge connecting nodes **i** and **j** is just the set {**i**, **j**} [14]. Throughout this document an edge is donated by $\mathbf{e_{ij}}$.

In a simple graph[3], an edge from node **A** to node **B** is considered to be the same thing as an edge from node **B** to node **A**. This graph is known as undirected graph. There are many other types of graphs. This section briefly defines the graphs that are refereed to in this project which are the directed graph, directed acyclic graph, and weighted graph.

Directed graph is a graph in which an edge from node **A** to node **B** is considered to be a distinct edge as an edge from node **B** to node **A**. Directed acyclic graph also called a DAG, is a special kind of directed graph with no directed cycles; that is, for any node **A**, there is no directed path that starts and ends on **A**. A graph is considered to be a weighted graph if a positive number i.e. weight is assigned to each edge. Such weights might represent costs, lengths, capacities, or anything else depending on the problem. Weight of the graph is the sum of the weights given to all edges.

The following section describes how the algorithm works.

4.3.2 Algorithm Description:

The algorithm is divided into two main parts which are *Direct Parent Finder* and *Indirect Parent Finder*. The *Direct Parent Finder* scans through the whole graph that is stored in the database and outputs two distinct graphs **O** and **T** from the initial graph. The graph **O** contains the nodes with one possible parent i.e. vertices that have only one edge to another vertex. This graph represents the algorithm final output. The graph **T** contains the nodes with multiple parents i.e. vertices that have more than one

[3]http://en.wikipedia.org/wiki/Graph$_($*mathematics*)

edge. Each node and its possible parents are represented as a separate graph within the graph **T**. The *Indirect Parent Finder* processes each graph in **T**, finds the parents for nodes that have more than one possible parent i.e. finds the eligible vertex for the vertices that have more than one edge, and adds the node and its eligible parent to **O**. Notice that the initial graph that is stored in the database is scanned only once. This is due to the fact that the initial graph could be a huge graph and processing it is a very heavy and resource consuming process.

Before describing each part in more detail, the variables and the thresholds that are used in the algorithm are introduced. The definitions of the variables that are used in the algorithm are listed below:

- **G** is the weighted directed graph that is stored in the database
- **N** is the number of vertices in **G**
- **V** is the set of all the vertices in **G**
- **E** is the set of all the edges in **G**
- **O** is a sub–graph of **G**. It is the final output of the algorithm.
- **T** is a sub–graph of **G** that the first part of the algorithm outputs. It is a temporary graph used to store the vertices that have more than one parent.

Moreover, the algorithm has two configurable parameters that are used as a measure to filter the graph from the noise. These parameters are the *Node Threshold* and the *Parent Threshold*. Both of the thresholds represent the value of the edge connected to a node. The *Node Threshold* determines when a node should be added to the final graph while the *Parent Threshold* determines when to consider a node as a parent to another node.

As mentioned in the previous section there is a dependency between the *Question Threshold* value and the *Node Threshold* value. The *Node Threshold* value should be equal to or less than the *Question Threshold*. The *Question Threshold* determines when an answer should be added as a question i.e. moving a concept to be a super–class or sub–class. The *Node Threshold* determines when a node i.e. class should be represented in the final graph. Logically speaking, all the nodes that were eligible to be moved as a super–class or sub–class should be represented in the final graph. If this is not the case, then gaps will be introduced in the final ontology. This means that the ontology will contain concepts that are not related, which is invalid as all the concepts in the ontologies produced by the *Ontology Refinement System* are meant to be related.

For example, lets assume that the *Node Threshold* is equal to 20 and the *Question Threshold* is equal to 15. The question displayed to the players is **"——- is a Kind of Animal?"**. Lets assume that 15 players agreed that **Mammal** is a kind of **Animal**. Therefore, **Mammal** is added as question. The new question is **"——- is a Kind of Mammal?"**. Lets assume that 20 players agreed that **Cat** is a kind of **Mammal**. Therefore **Cat** is added as question. This information is added to the game's database. When the algorithm processes the graph stored in the database, the algorithm will add the **Cat** node to the final graph as the value of its edge is greater than the *Node Threshold*. However, the algorithm will not add the **Mammal** node to the final graph as the value of its edge is less than the *Node Threshold* creating gap in the final ontology.

The following sections describe the *Direct Parent Finder* and the *Indirect Parent Finder* in more detail.

Direct Parent Finder

The *Direct Parent Finder* which is the first part of the algorithm, aims to find the nodes in the initial graph **G** that have one direct parent and add them to the final graph **O**. Moreover, it adds all the other nodes that have more than one direct parent to the temporary graph **T**. It is the only place where the initial graph **G** is scanned completely. It takes **G** as an input, processes it, and outputs two graphs **O** and **T**. Both **O** and **T** are sent to the *Indirect Parent Finder* for further processing. The following is a step–by–step description of what the *Direct Parent Finder* does:

a. For each node in **G** find the set of all parents. In other words, for each vertex find the set of all the edges and the set of all the vertices that are directly linked to that vertex.

   So, for each vertex $\mathbf{v_i}$ that belongs to the set of vertices **V** such that **i** is less than or equal to the number of vertices **N**, find all the edges $\mathbf{E_{c(i)}}$ of $\mathbf{v_i}$ such that $\mathbf{E_{c(i)}}$ belongs to **E** and the set of vertices $\mathbf{V_j}$ such that $\mathbf{V_j}$ is a subset of **V** and there exists an $\mathbf{e_{ij}}$ such that $\mathbf{e_{ij}}$ is the direct link between $\mathbf{v_i}$ and $\mathbf{v_j}$ and $\mathbf{e_{ij}}$ belongs to $\mathbf{E_{c(i)}}$ and $\mathbf{v_j}$ belongs to $\mathbf{V_j}$.

   Note that $\mathbf{v_i}$ represents the sub–class or child node and $\mathbf{V_j}$ represents the set of all possible parents of $\mathbf{v_i}$.

b. If the node has no associated parents i.e. the set $\mathbf{E_{c(i)}}$ is empty, then this node is the root node. So, add it to the final graph **O** as a root node and go back to step a.

c. If the node has one or more parents, then make sure that this node is not a noise i.e. it is eligible to be represented in the final graph.

   A vertex is eligible to be represented in the final graph if it is connected to an edge such that the weight of the edge is greater than a threshold. The threshold is used to filter the noise, improper data from the input graph, and remove the vertices that should not be represented in the output. This can be done as follows:

   If there exists $\mathbf{e_{ij}}$ such that $\mathbf{e_{ij}}$ belongs to $\mathbf{E_{c(i)}}$ and $\mathbf{e_{ij}}$ has a weight greater than or equal to the *Node Threshold* then the vertex $\mathbf{v_i}$ is eligible to be represented in the output graph **O**, so proceed to the next step otherwise go back to step a.

d. After filtering the node i.e. sub–class, the set of possible parents should be filtered as well. Only the eligible parents should be considered in the list of possible parents.

   A vertex is eligible to be considered in the list of possible parents, if it is connected to an edge such that the weight of the edge is greater than a threshold. This is used to filter the noise, improper data from the graph, and remove the vertices, parents, which should not be considered while processing the sub–classes. This can be done as follows:

   If there exists only one $\mathbf{e_{ij}}$ such that $\mathbf{e_{ij}}$ belongs to $\mathbf{E_{c(i)}}$ and $\mathbf{e_{ij}}$ has a weight greater than or equal to the *Parent Threshold* then the vertex $\mathbf{v_j}$ is the only eligible parent so add $\mathbf{v_i}$ and $\mathbf{v_j}$ to the output graph $\mathbf{O}$ and go back to step a.

   If there is more than one edge $\mathbf{e_{ij}}$ that has a weight greater than or equal to the *Parent Threshold*, then add all the $\mathbf{v_j}$ that are connected by these $\mathbf{e_{ij}}$ to the set of temporary parent vertices $\mathbf{V_{tp}}$ and proceed to the next step.

   Note that the value of the *Parent Threshold* should be less than the value of the *Node Threshold*. The reason for this is to enable specialisation of concepts i.e. moving the nodes from general to specific parent by reducing the value of the *Parent Threshold* but keeping it sufficiently high to filter the noise i.e. improper parents.

e. After finding the set of temporary parents, the relationship between these parents should be found i.e. whether a parent is a sub–class of another parent. This helps in finding the most accurate specialised parent. It also helps in eliminating the explicit redundant data which is discussed later.

   The relationship is found through the Cartesian–Product of all the temporary parents.

   Let **CP** be the set of Cartesian–Product such that

   $$\mathbf{CP} = \mathbf{V_{tp}} \times \mathbf{V_{tp}} = \{(v_\mathbf{a}, v_\mathbf{b}) | v_\mathbf{a}, v_\mathbf{b} \in V_\mathbf{tp}, v_\mathbf{a} \neq v_\mathbf{b}\}$$

f. After finding the Cartesian–Product of the temporary parents, the parents that are not related are removed from the set of the Cartesian–Product. The parents that are not related can be found as follow:

   If the weight of the edge from **A** to **B** is greater than zero and the weight of the edge from **B** to **A** is zero, then there is no relation between **B** and **A**, but there is a relation between **A** and **B**.

$$\mathbf{A} \xrightarrow{>0} B$$

$$\mathbf{A} \xleftarrow{=0} B$$

Note that if the weight of the edge in both directions is equal to zero, then the relationship between the two parents can not be determined. This is because the case of a sub–class having more than one parent is taken into account. In this case the parents are not related to each other but they are still valid parents. So, they should be among the other elements in the Cartesian–Product in order to be considered as parents in later stages.

In other words, for each element **(x,y)** in the **CP** list, if there exists **(a, b)** such that $\mathbf{e_{ab}}$ exists but $\mathbf{e_{ba}}$ does not exists, then remove **(b, a)** from **CP**.

g. After removing the non–related elements from the Cartesian–Product of the temporary parents, if there is only one element in the Cartesian–Product, then the value in the left side of the element is the parent e.g. **x** is the parent in the element **(x, y)**. This is because the left value is more specific than the right one as **(x, y)** is interpreted as **x** is a sub–class of **y**. The parent and its child are added to the final graph, and the algorithm goes back to step a.

   If there is more than one element in the Cartesian–Product, then these elements are added to the temporary parent's graph with its associated child, and the algorithm goes back to the step a.

h. The whole graph is scanned once. If there are nodes in the temporary graph the algorithm executes the second part to find the parents for the nodes that still have many possible parents.

The Pseudo–Code of the above algorithm is illustrated in algorithm 4.1.

Algorithm 4.1: Direct Parent Finder

```
Input: $G$ [ a weighted directed graph with $N$ vertices]
Data: $N$ is the number of vertices in $G$
      $V$ is the set of all the vertices in $G$
      $E$ is the set of all the edges in $G$
      $O$ is a sub–graph of $G$. It is the final output of the algorithm.
      $T$ is a sub–graph of $G$ that the first part of the algorithm outputs. It is a
      temporary graph used to store the vertices that have more than one parent
      $NT$ is the NODE THRESHOLD
      $PT$ is the PARENT THRESHOLD
Output: $O$ and $T$

$O \longleftarrow \emptyset$
$T \longleftarrow \emptyset$
$NT \longleftarrow 50$
$PT \longleftarrow 20$

foreach ($v_i$ in the set $V$, such that $i <= N$) do
    Set $E_{c(i)}$ to be the set of all the edges of $v_i$ such that $E_{c(i)} \in E$
    /* Note $v_i$ represents a sub--class                                    */
    Set $V_j$ to be the set of all the vertices such that:
      $\exists\ e_{ij} \in E_{c(i)}$ such that:
          $e_{ij}$ is the immediate links between $v_i$ and $v_j$, $v_j \in V_j$ and, $V_j \sqsubseteq V$
      /* Note $V_j$ represents the set of super classes i.e.  parents of a
         particular subclass $v_i$                                           */
    if ($E_{c(i)} = \emptyset$) then
        add $v_i$ as a root vertex in $O$
    else if ($\exists\ e_{ij} \in E_{c(i)}$ such that $e_{ij} >= NT$) then
        if ($\exists{=}1\ e_{ij} \in E_{c(i)}$ such that $e_{ij} >= PT$) then
            add $v_i$ and $v_j$ to $O$ such that $v_j$ is an endpoint of $e_{ij}$
        else if ($\exists{>}1\ e_{ij} \in E_{c(i)}$ such that $e_{ij} >= PT$) then
            add all the $v_j$ that are connected by these $e_{ij}$ to the set of temporary
            parent vertices $V_{tp}$
            Let $CP$ be the set of Cartesian Product such that :
            $CP = V_{tp} \times V_{tp} = \{(v_a, v_b) | v_a, v_b \in V_{tp}\ ,\ v_a \neq v_b\ \}$
            foreach ($(x, y) \in CP$) do
                if ($\exists\ (a, b)$ such that ($\nexists\ e_{ab}$) and ($\exists\ e_{ba}$)), $e_{ab}$, $e_{ba} \in E_{c(i)}$ then
                    remove $(a, b)$ from $CP$ such that:
                        $e_{ab}$ is the immediate link between $a$ and $b$
                        $e_{ba}$ is the immediate link between $b$ and $a$
                end
                if ($\exists{=}1 (x, y) \in CP$) then
                    add $v_i$ and $x$ to $O$
                else if ($\exists{>}1\ (x, y) \in CP$) then
                    add $v_i$ and $CP$ to $T$
                end
            end
        end
    end
end
```

Indirect Parent Finder

The *Indirect Parent Finder* which is the second part of the algorithm, takes **O** and **T** as input and outputs **O**. The main functionality of the second part is to find the most accurate parent for the node that has more than one parent. The node and its parents are stored in **T**. **T** actually contains **N** number of sub–graphs that are isolated. Each sub–graph consists of a node and its associated parents.

So, for each sub–graph that belongs to **T**, the sub–graph is sent to a recursive function **fp**. This function finds the parent of the node in the sub–graph. The function is able to detect and remove any cycle across the sub–graphs that are sent to it. The function **fp** returns false if a cycle is found and true if the parent is found. So, in order to make sure that all the cycles are removed and a parent is found for the node in each sub–graph, the algorithm keeps calling **fp** until all the cycles are removed and a parent is found.

Due to the fact that the algorithm keeps calling the function in an internal loop and the function is a recursive function, the direct parent of some other nodes might be found while processing a node. In order to prevent the processing of the same node again, a check has been added before calling the loop to verify if the parent of that node has been already found.

Let $(\mathbf{t_i},\ \mathbf{CP_i})$ be a sub–graph in **T** such that **i** is less than **A**, $\mathbf{t_i}$ is the node i.e. sub–class that has more than one parent, $\mathbf{CP_i}$ is the set of all the possible parents for $\mathbf{t_i}$, and **A** is the number of sub–graph in **T**. The Pseudo–Code of the *Indirect Parent Finder* algorithm is illustrated in algorithm 4.2.

Algorithm 4.2: Indirect Parent Finder

```
Input: $T$ and $O$
Data: $CP$ is a sub–graph in $T$
Output: $O$
Require: $T \neq \emptyset$
foreach $((t_i, CP_i) \in T)$ do
    if $(t_i \notin O)$ then
        /* If a cycle is found and removed fp returns false, so process the
           same graph again as the current processing has been rolled back
           */
        while (! fp $(t_i, CP_i)$) do
            fp $(t_i, CP_i)$
        end
    end
end
```

The following section is a step–by–step description of how the function **fp** works:

The Recursive Function fp:

The function **fp** takes a graph as an input and returns true or false. The only variable required by the function is **C**.

Let **C** be a sub–graph of **T**. **C** is used to trace cycles. Initially **C** is an empty graph.

The function works as follow:

Each parent in the list of possible parents is used to eliminate the possible parents which leads to finding the most accurate parent. In order to do this, the parents' hierarchy of each parent should be known.

a. For each element $(\mathbf{x}, \mathbf{y})$ that belongs to $\mathbf{CP_i}$ such that $(\mathbf{t_i}, \mathbf{CP_i})$ belongs to **T**, perform the following:

b. First make sure that there is no cycle so far. This can be done by verifying that the algorithm was not trying to find the parent of $\mathbf{t_i}$ before.

   If this is the case, then a cycle is found. In order to remove the cycle, the algorithm finds the smallest weighted edge that is involved in the cycle and removes it. After this the algorithm clears **C** and returns false.

c. If there is no cycle, then verify whether the direct parent of **x** is known. This can be done by checking if the parent **x** exists in the final graph **O**.

   If the parent **x** does not exist in the final graph **O**, then the algorithm adds $\mathbf{t_i}$ to **C** in order to keep track of all the vertices that have been processed recursively to trace a cycle. Then the algorithm calls the same **fp** function again passing the parent **x** as the vertex in which a parent needs to be determined. If the function returns false i.e. a cycle has been removed, then the processing of all the vertices that are involved in the cycle is rolled back. This is because of the fact that the element that has been removed from the original possible parents $\mathbf{CP_i}$ might belong to any sub–graph in **T** such that its vertex $\mathbf{t_i}$ is involved in the cycle and this might affect the accuracy of the vertices that are involved in the cycle and have been processed partially.

d. If the parent **x** exists in the final graph **O**, and if there exists only one element in $\mathbf{CP_i}$, then add the left value as a parent to $\mathbf{t_i}$ in the final graph **O**, remove the processed graph $(\mathbf{t_i}, \mathbf{CP_i})$ from the temporary graph **T**, and return true.

e. If there is more than one element in $\mathbf{CP_i}$, then find the parent's hierarchy of **x** and store it as a graph in **SP** such that **x** is the lowest node in the graph. The order of the element in **SP** is important as it shows the hierarchy of the parents starting from the direct parent of $\mathbf{t_i}$.

   The parent's hierarchy is used to remove the redundant data by first removing the connected parents and then removing the indirect parents from the list of possible parents $\mathbf{CP_i}$.

f. Remove all the connected parents from the list of possible parents $\mathbf{CP_i}$. This can be done by iterating through the parent's hierarchy and for each element **a**, if the element **a** exists in the list of possible parents $\mathbf{CP_i}$ as both a child element and a parent element, then remove all the elements from $\mathbf{CP_i}$ where **a** is a child element. This actually removes the redundant information and results in the identification of the most specialised parent.

For example suppose,
SP = (5, 9, 1, 3) , and
$\mathbf{CP_i} = \{(5, 1), (5, 3), (8, 5)\}$

After removing the connected parent,
$\mathbf{CP_i} = \{(8, 5)\}$

g. If there exists only one element in $\mathbf{CP_i}$, then add the left value as a parent to $\mathbf{t_i}$ in the final graph **O**, remove the processed graph $(\mathbf{t_i}, \mathbf{CP_i})$ from the temporary graph **T**, and return true.

h. Remove all the indirect parents from the list of possible parents $\mathbf{CP_i}$. This can be done by iterating through the parent's hierarchy and for each two nearest parents **a** and **b** that are found in $\mathbf{CP_i}$, remove all the other elements $(\mathbf{x}, \mathbf{y})$ from $\mathbf{CP_i}$ such that **a** is equal **x** but **b** is not equal to **y**.

For example suppose,
SP = (1, 6, 2, 8), and
CPi = {(1, 2), (1, 8), (6, 2)}

After removing the indirect parents,
CPi = {(1, 2), (6, 2)}
(1, 8) is implicitly known.

i. If there exists only one element in $\mathbf{CP_i}$, then add the left value as a parent to $\mathbf{t_i}$ in the final graph **O**, remove the processed graph $(\mathbf{t_i}, \mathbf{CP_i})$ from the temporary graph **T**, and return true.

j. If there exists more than one element in $\mathbf{CP_i}$ and **x** was the last parent to be processed in the current graph $(\mathbf{t_i}, \mathbf{CP_i})$, then in this case the algorithm concludes that the element $\mathbf{t_i}$ has more that one parent. Therefore, add each left value in $\mathbf{CP_i}$ as parent to $\mathbf{t_i}$ in the final graph **O**, remove the processed graph $(\mathbf{t_i}, \mathbf{CP_i})$ from the temporary graph **T**, and return true.

k. If there exists more than one element in $\mathbf{CP_i}$ and **x** was not the last parent to be processed in the current graph $(\mathbf{t_i}, \mathbf{CP_i})$, then in this case the algorithm goes back to step a.

The Pseudo–Code of the Recursive Function *fp* is illustrated in algorithm 4.3.

Algorithm 4.3: Recursive Function fp

```
Input: $t_{\mathrm{i}}$ and $CP_{\mathrm{i}}$
Data: $O$ is a sub–graph of $G$. It is the final output of the algorithm.
      $T$ is a sub–graph of $G$ that the first part of the algorithm outputs. It is a
      temporary graph used to store the vertices that have more than one parent
      $C$ is a sub–graph of $T$. $C$ is used to trace cycles
Output: true or false
$C \longleftarrow \emptyset$
if ($t_i \in C$) then
    find $e_{\mathrm{ab}}$ such that $e_{\mathrm{ab}} \in C$ and $e_{\mathrm{ab}}$ is the smallest edge
    remove $(a, b) \in (t_{\mathrm{i}}, CP_{\mathrm{i}})$ such that:
      $(t_{\mathrm{i}}, CP_{\mathrm{i}}) \in T$, $b$ is the endpoint of $e_{\mathrm{ab}}$ and $a$ is the start point of $e_{\mathrm{ab}}$
    set $C = \emptyset$
    return false
end
foreach (($(x, y) \in CP_i$, such that $(t_i, CP_i) \in T$) do
    if ($x \notin O$) then
        add $t_{\mathrm{i}}$ to $C$
        fp ($x, CP_{\mathrm{i}}$)
    else
        if ($\exists{=}1(x, y) \in CP_i$) then
            add $(t_{\mathrm{i}}, x)$ to $O$ such that $(t_{\mathrm{i}}, x) \in T$
            remove $(t_{\mathrm{i}}, CP_{\mathrm{i}})$ from $T$ such that $(t_{\mathrm{i}}, CP_{\mathrm{i}}) \in T$
            return true
        end
        Let $SP$ be the set of all the vertices $v$ such that:
          $\exists$ $e_{\mathrm{ij}}$ from $v_{\mathrm{i}}$ to $v_{\mathrm{j}}$, $v_{\mathrm{i}}$ and $v_{\mathrm{j}} \in$ O AND $v_0 = x$
        remove $\forall (a, b) \in CP_{\mathrm{i}}$ such that:
          $\exists$ $e_{\mathrm{ab}} \in T$,
          $\exists$ $(c, d) \in CP_{\mathrm{i}}$,
          $a = d$, and $a \in SP$
        if ($\exists{=}\ 1(x, y) \in CP_i$) then
            add $(t_{\mathrm{i}}, x)$ to $O$ such that $(t_{\mathrm{i}}, x) \in T$
            remove $(t_{\mathrm{i}}, CP_{\mathrm{i}})$ from $T$ such that $(t_{\mathrm{i}}, CP_{\mathrm{i}}) \in T$
            return true
        end
        remove $\forall$ $(x, y) \in CP_{\mathrm{i}}$ such that:
          $\exists$ $e_{\mathrm{xy}} \in T$,
          $x = a$,
          $y \neq b$,
          $a$, $b \in SP$, and
          $a$ and $b$ are the nearest elements in the list that exists in $CP_{\mathrm{i}}$
        if ($\exists{=}\ 1(a, b) \in CP_i$) then
            add $(t_{\mathrm{i}}, a)$ to $O$ such that $(t_{\mathrm{i}}, a) \in T$
            remove $(t_{\mathrm{i}}, CP_{\mathrm{i}})$ from $T$ such that $(t_{\mathrm{i}}, CP_{\mathrm{i}}) \in T$
            return true
        else if ($\exists{>}\ 1(a, b) \in CP_i$ AND $x$ is the last element of $CP_i$) then
            foreach (($(a, b) \in CP_i$) do
                add $(t_{\mathrm{i}}, a)$ to $O$ such that $(t_{\mathrm{i}}, a) \in T$
            end
            remove $(t_{\mathrm{i}}, CP_{\mathrm{i}})$ from $T$ such that $(t_{\mathrm{i}}, CP_{\mathrm{i}}) \in T$
            return true
        end
    end
end
```

4.3.3 Example

This section provides an example that demonstrates a step–by–step how the algorithm works. The example covers almost all the different cases that the algorithm handles.

Assume that figure 4.4 represents a collective assertion of the players' answers in isolated game sessions for an ontology **G**. Assume that the *Parent Threshold* is set to 10 and the *Node Threshold* is set to 20.

Note that the algorithm is not affected by the order in which the elements are processed. It always gives the same answer. The algorithm has been tested with different scenarios and each time it gave the same answer.

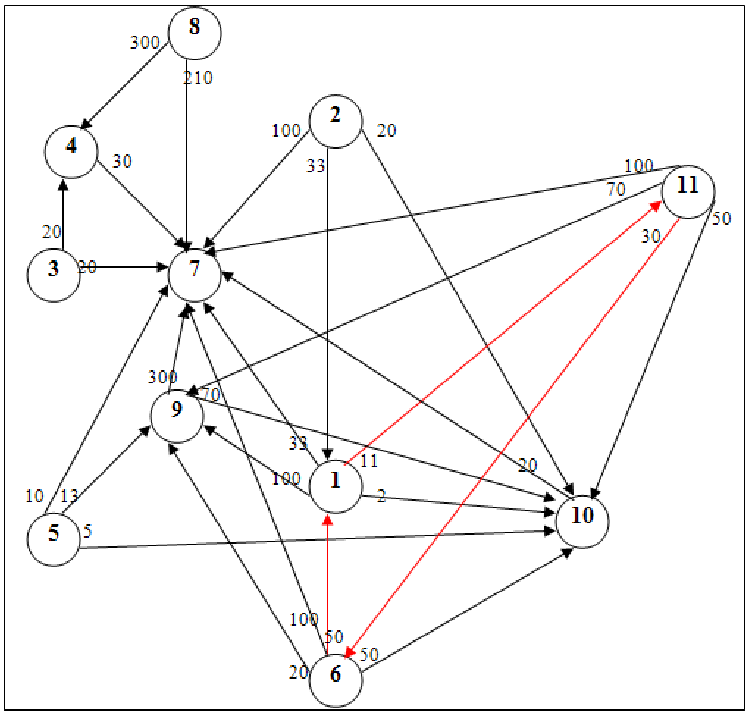


Figure 4.4: Example Input

Direct Parent Finder

Case 1: Root Element
Let $\mathbf{v_i}$ be the vertex with label 7
In this case $\mathbf{E_{c(i)}}$ is $\emptyset$ as there is no edges connected from the vertex
Therefore $\mathbf{V_j}$ is also $\emptyset$
As stated in the algorithm

```
if ($E_{c(i)} = \emptyset$) then
  add $v_i$ as a root vertex in $O$
end if
```

Vertex with label 7 is the root element.
The vertex 7 is added to the final graph **O** as a root element.

Case 2: One Direct Parent
Let $\mathbf{v_i}$ be the vertex with label 3
In this case $\mathbf{E_{c(i)}}$ = { $\mathbf{e_{3,4}}$, $\mathbf{e_{3,7}}$ } and $\mathbf{V_j}$ = {4, 7}
$\mathbf{E_{c(i)}}$ is not an empty set. Therefore, 3 is not a root element

$\mathbf{e_{3,4}} = 20$ and $\mathbf{e_{3,7}} = 20$, therefore there exists an edge such that its value is greater than or equal to the *Node Threshold*, which is 20

In this case, both of the edges are greater than the *Parent Threshold*, so there exist more than one edge that is greater than or equal to the *Parent Threshold*, so
$\mathbf{V_{tp}} = \{4, 7\}$
$\mathbf{CP_i} = \mathbf{V_{tp}} \times \mathbf{V_{tp}} = \{(4, 7), (7, 4)\}$
$\mathbf{e_{e4,7}} = 30$ and,
$\mathbf{e_{e7,4}} = 0$ which means it does not exist

As stated in the algorithm if there exist (**a**, **b**) such that ($\nexists\ e_{\mathrm{ab}}$) and ($\exists\ e_{\mathrm{ba}}$)) then remove (**a**,**b**)

Therefore, the element (7, 4) is removed from $\mathbf{CP_i}$
$\mathbf{CP_i} = \{(4, 7)\}$

Because, there exist only one element in $\mathbf{CP_i}$, then the left element, which is 4, is the parent of $\mathbf{v_i}$ which is 3

3 and 4 are added to the final graph **O**

In the same manner the other vertices are processed. The output of the first part of the algorithm is as follow:
Vertex 1 and its **CP** = {(9,7), (11,7), (11,9)} are added to **T**
Vertex 2 and its **CP** = {(1,7), (10,7), (1,10)} are added to **T**
Vertex 6 and its **CP** = {(1,7), (9,7), (10,7), (1,9), (1,10), (9,10)} are added to **T**
Vertex 11 and its **CP** = {(6,7), (9,7), (10,7), (6,9), (6,10), (9,10)} are added to **T**

Vertex 4 and its parent 7 are added to **O**
Vertex 8 and its parent 4 are added to **O**
Vertex 9 and its parent 10 are added to **O**
Vertex 10 and its parent 7 are added to **O**

Vertex 5 is not added to any of the graphs as it does not have any edge that is greater than or equal to the *Node Threshold*.

The output of the *Direct Parent Finder* is illustrated in figure 4.5 and figure 4.6.

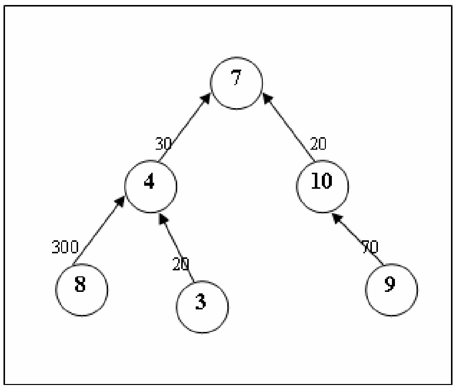


Figure 4.5: Direct Parent Finder Output (Graph O)

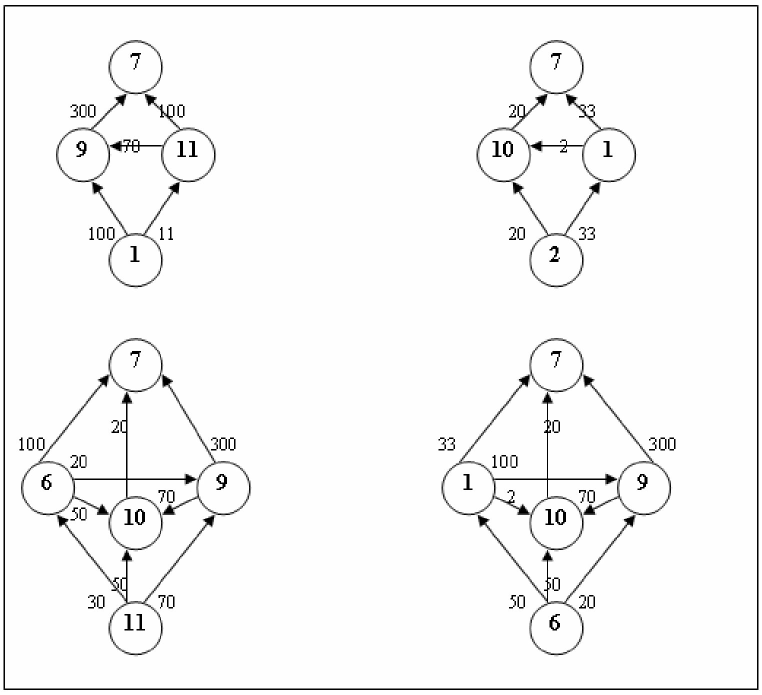


Figure 4.6: Direct Parent Finder Output (Graph T)

Indirect Parent Finder

In the second part of the algorithm, each sub–graph of **T** enters a loop that keeps calling the **fp** function until a parent is found.

<u>**Case 3:**</u> Cycle
Starting with the first sub–graph in **T**,
let $\mathbf{t_i}$ be the vertex with label 1 and $\mathbf{CP_i}$ = {(9,7), (11,7), (11,9)}
$\mathbf{t_i}$ does not exist in **O**
Let $\mathbf{C} = \emptyset$

1 does not exist in **C**
Loop through each element in CP$_i$

1. Let (**x**, **y**) = (9, 7)
   9 exists in **O**
   CP$_\mathbf{i}$ has more than one element, so proceed
   SP = (9, 10, 7)

   Removing connected parents:
   Let (**a**, **b**) = (9, 7)
   Let (**c**, **d**) = (11, 9)
   a = **d** = 9 and **a** exists in **SP**, so remove (9, 7) from **CP$_\mathbf{i}$**
   CP$_\mathbf{i}$ = (11,7), (11,9)

   Still more than 1 parent exits, so proceed

   Removing indirect parents:
   There is no indirect parent to remove

   Still more than 1 parent exits, so proceed with another element in **CP$_\mathbf{i}$**

2. Let (**x**, **y**) = (11, 7)
   11 does not exist in **O**
   t$_\mathbf{i}$ which is 1 does not belong to **C**
   So, 1 is added to **C**
   C = {1}

   Internal call to fp:
   Let **t$_\mathbf{i}$** be the vertex with label 11,
   and **CP$_\mathbf{i}$** = {(6,7), (9,7), (10,7), (6,9), (6,10), (9,10)}
   11 does not exist in **C**

   Loop through each element in CP$_i$

   (a) Let (**x**, **y**) = (6, 7)
       6 does not exist in **O**
       11 does not belong to **C**
       So, 11 is added to **C**
       C = {1, 11}

       Internal call to fp:
       Let **t$_\mathbf{i}$** be the vertex with label 6,
       and **CP$_\mathbf{i}$** = {(1,7), (9,7), (10,7), (1,9), (1,10), (9,10)}
       6 does not exist in **C**

       Loop through each element in CP$_i$

       i. Let (**x**, **y**) = (1, 7)
          1 does not exist in **O**
          6 does not belong to **C**

So, 6 is added to **C**
C = {1, 11, 6}

Internal call to fp:
Let $\mathbf{t_i}$ be the vertex with label 1,
and $\mathbf{CP_i}$ = {(9,7), (11,7), (11,9)}
1 exists in **C**

Removing the cycle:
C = {1, 11, 6} and,
$\mathbf{e_{1,11}} = 11$,
$\mathbf{e_{11,6}} = 30$,
$\mathbf{e_{6,\,1}} = 50$,
$\mathbf{e_{1,11}}$ is the smallest edge, therefore the element (1, 11) is removed from the sub–graph that belongs to **T** and has 1 as the $\mathbf{t_i}$ vertex.

C = ∅
Return false

All the internal calls of the function **fp** return false, so proceed with the main loop in the second part of the algorithm that calls the **fp** function again with the same sub–graph that returned false.

Figure 4.7 represents the graph **T** after removing the cycle.

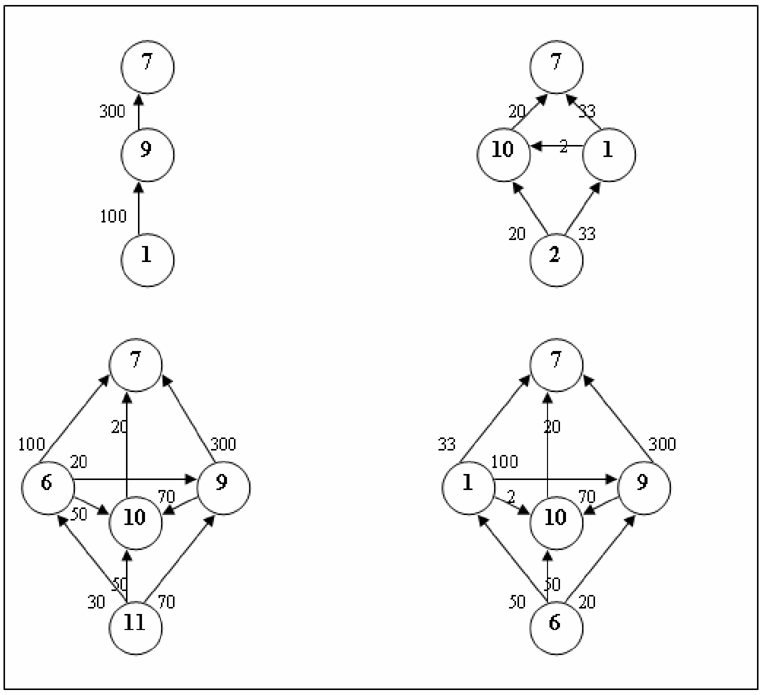


Figure 4.7: Indirect Parent Finder Removes Cycle From Graph T

Starting again with the first sub–graph in **T**,
let $\mathbf{t_i}$ be the vertex with label 1 and $\mathbf{CP_i}$ = {(9,7)}
1 does not exist in **O**

Let **C** = ∅

1 does not exist in **C**
Loop through each element in $\boldsymbol{CP_i}$

1. Let (**x**, **y**) = (9, 7)
   9 exists in **O**
   $\mathbf{CP_i}$ has only one element so,
   add 1 and its parent 9 to the final graph **O**
   remove 1 and its $\mathbf{CP_i}$ from **T**
   Return true

Figure 4.8 and figure 4.9 represent the new graphs (**T** and **O**) after processing vertex 1.

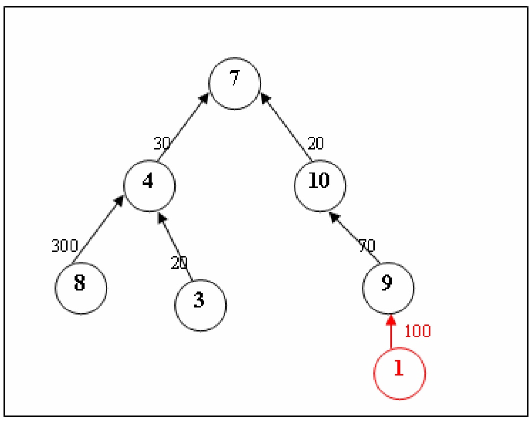


Figure 4.8: Indirect Parent Finder Outputs Graph O after Processing Vertex 1

Processing the second sub–graph in **T**,
Let $\mathbf{t_i}$ be the vertex with label 2 and $\mathbf{CP_i}$ = {(1,7), (10,7), (1,10)}
2 does not exist in **O**
Let **C** = ∅

2 does not exist in **C**
Loop through each element in $\boldsymbol{CP_i}$

1. Let (**x**, **y**) = (1, 7)
   1 exists in **O**
   $\mathbf{CP_i}$ has more than one element, so proceed
   SP = (1, 9, 10, 7)

   Removing linked parents:
   Let (**a**, **b**) = (10, 7)
   Let (**c**, **d**) = (1, 10)
   a = **d** = 10 and **a** exists in **SP** so remove (10, 7) from $\mathbf{CP_i}$
   $\mathbf{CP_i}$ = {(1,7), (1,10)}

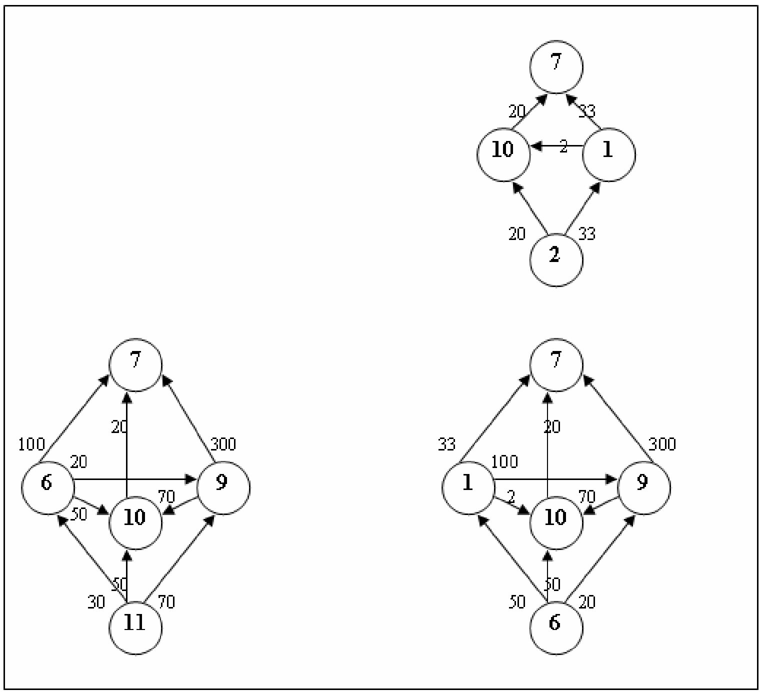


Figure 4.9: Indirect Parent Finder Outputs Graph T after Processing Vertex 1

Still more than 1 parent exits, so proceed

Removing indirect parents:
For each element in **SP**, find the element and its nearest parent in the list such that it exists in $\mathbf{CP_i}$

For element 1 in **SP**, the nearest parent that can be found in $\mathbf{CP_i}$ is 10
So, remove from $\mathbf{CP_i}$ all the elements ($\mathbf{x}$, $\mathbf{y}$) such that $\mathbf{x} = 1$ and $\mathbf{y} \neq 10$
The element (1, 7) is removed from $\mathbf{CP_i}$

There exits only one element in $\mathbf{CP_i}$ so,
add 2 and its parent 1 to the final graph **O**
remove 2 and its $\mathbf{CP_i}$ from **T**
Return true

In the same manner the other vertices (6 and 11) are processed. Figure 4.10 represents the final graph **O** after processing all the sub–graphs in **T**.

4.4 Ontology Representation Generator Component

The *Ontology Representation Generator Component* is responsible for representing the output of the *Ontology Builder Component* formally. The output of the *Ontology Builder Component* is a graph that represents an ontology. The *Ontology Representation Generator Component* takes the ontology and represents it in any ontology language that is specified in the component. The ontology language that is used in this

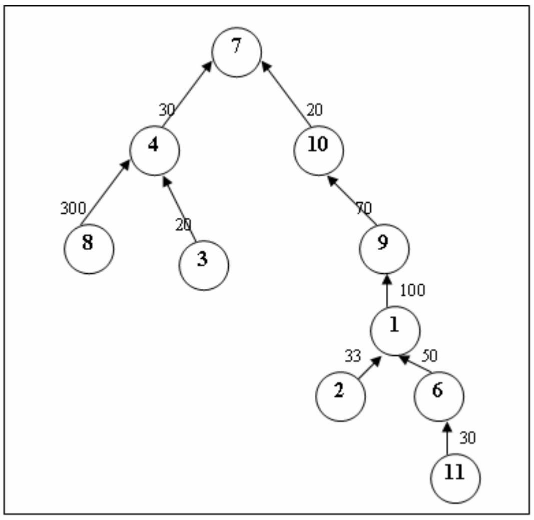


Figure 4.10: Indirect Parent Finder Final Output (Graph O)

thesis is the OWL–DL.

The *Ontology Builder Component* processes the knowledge collected by the *Game Component* and produces a graph. This graph is sent to the *Ontology Representation Generator Component* as a set of statements. The *Ontology Representation Generator Component* processes each statement and represents it in an OWL–DL format with a “is-sub-class-of” relationship between the nodes in the statement. If the statement has only one node, then it infers that the node in the statement is a root node. Thus, it adds the comment that states it is a root node to this node and it does not add the relationship “is-sub-class-of” to this node. The *Ontology Representation Generator Component* uses the ontology name to define a unique namespace[4] for the ontology and its nodes. The *Ontology Representation Generator Component* produces a physical file representing each ontology.

[4]A Namespace uniquely identifies a set of names so that there is no ambiguity when objects having different origins but the same names are mixed together.

Chapter 5

Implementation

This chapter provides technical details of the implementation of the *Ontology Refinement System*. It presents the programming languages, frameworks, and tools that have been used during the development of each component in the system. Moreover, it provides screen shots of the *Game Component* interface and a sample output file of the *Ontology Representation Generator Component*. This chapter explains the internal representation of the data structure that has been used in the *Ontology Builder Component*. Furthermore, it describes the technical limitations of the *Ontology Refinement System*.

5.1 Main System Architecture

The *Ontology Refinement System* has been implemented purely in Java. Some of the system's components have been deployed in a Web server so that they can be accessed publicly through the Internet or a network, while other components have been deployed in a game server to prevent direct access to them. These components provide features that serve the game and the ontology generation functionalities. Figure 5.1 illustrates the main system architecture.

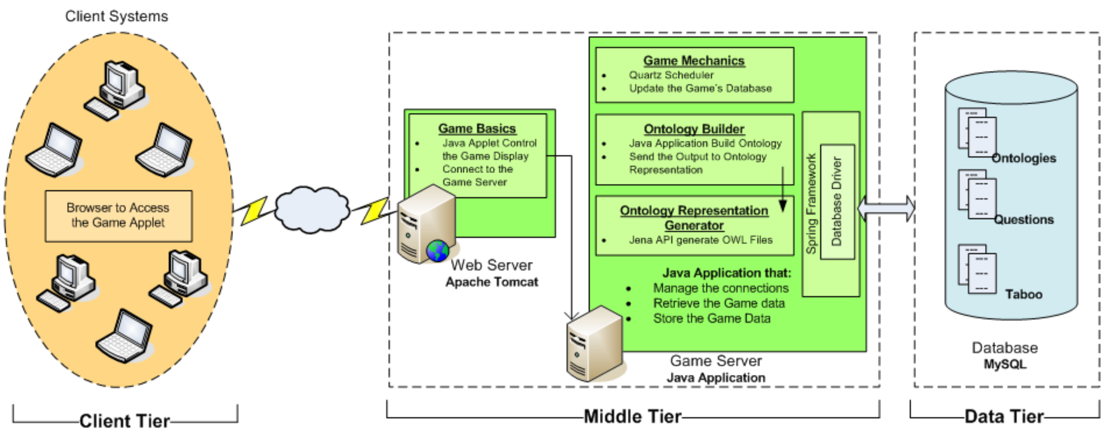


Figure 5.1: Ontology Refinement System Architecture

The *Ontology Refinement System* is implemented in three different tiers. The first tier is the client tier. It represents the players' systems. The players can access the game through their Internet browsers. The second tier is the middle tier. It contains the main system's components which are the *Game Component*, *Ontology Builder Component*, and the *Ontology Representation Generator Component*. It has two servers which are the Web server and game server. The Web server enables the players to access the game through a URL. On the other hand, the game server is responsible for managing all the logged in players and live game sessions. It initialises the game when there are two free players and stores the game information when the game is over. Moreover, it receives events from the Web server, processes them, and sends the response back to the Web server. Examples of the events are log in, answer, pass, and time over. The last tier is the data tier. This tier is responsible of storing the data required to start a game and the data collected at the end of every game.

5.2 Technologies

There are different frameworks and tools that have been used in order to develop the *Ontology Refinement System's* components. This section briefly introduces the tools that have been used.

5.2.1 Java Applet

An applet[1] is a program written in the Java programming language. It can be included in an HTML page. It aims to provide interactive features to Web applications which cannot be provided by plain HTML. Applets are compatible to be executed by browsers in any platform such as Windows and Linux.

The *Game Basics* subcomponent has been implemented as a Java Applet. This is due to the fact that the game should be interactive and accessible globally through the Internet or locally in a network. The *Game Basics* subcomponent is deployed in a Web server and it connects to a game server. More information about the Web server and game server is provided later in this section.

The *Game Basics* subcomponent is responsible for displaying the game to the players, sending events to the game server, receiving events from the game server and updating the game interface based on the game server response.

5.2.2 Web Server

The game should be accessible through the Internet or a network. This can be accomplished through a Web server. A Web server is a software that receives requests from Web browsers and responds back with the requested resources. The resources are mostly in the form of Web pages. The Web server should be installed in a machine that is accessible through the Internet or a network to enable the clients to request for resources.

[1]http://java.sun.com/applets/

Apache Tomact 5.5[2] Web server has been installed in order to make the game available through the Internet or a network. Tomact has been used because it is an open source software that most of the developers use to deploy their Java applications. It has many tools that can facilitate managing and configuring the deployed applications. The *Game Basics* subcomponent is deployed in Tomcat.

5.2.3 Game Server

The game server is a Java application that has been developed by the author to act as a server. The game server provides different features to the *Ontology Refinement System.* It connects to the *Game Basics* subcomponent through socket [3] connection. This connection is used to receive events from the *Game Basics* subcomponent and send the responses back to the subcomponent. The events include new player logs in, question pass request, new answer to a question, and time out. In addition, the game server handles multiple game sessions. It starts a new game session when two free players are available, matches the answers provided by two players playing in the same game session, allows new question to be displayed if the two players playing in the same game session agreed to pass a question, and frees the players when the game session is over. Moreover, the game server connects to the database through Spring framework to retrieve the game initial data and store the knowledge collected from every game session. The *Game Mechanics* subcomponent, *Ontology Builder Component*, and *Ontology Representation Generator Component* are deployed in the game server.

The initial code of the game framework has been taken from [6]. This code has the socket connectivity implementation and some simple events like, logging in, logging out, and chatting. The client interface in the code provided by [6] is a command line. The implemented game is capable of managing one session at a time i.e. two players playing together.

5.2.4 Quartz Scheduler

Scheduler is a program that executes a specific task which is a program as well in a specific time pattern e.g. daily or monthly or after a period of time. Quartz Scheduler[4] is an open source job scheduling system that can be integrated with any Java application.

The *Game Machines* subcomponent is integrated with the Quartz Scheduler to implement a scheduled job. This scheduled job is responsible for updating the question and taboo lists in the game's database. The implementation of the Quartz Scheduler has been accomplished through the integration classes that the Spring framework offers. The job runs independently of the game. It updates the data in the database so that future rounds of the game can retrieve the updated questions. The job runs every 10 minutes.

[2]http://tomcat.apache.org/

[3]A server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request

[4]http://www.opensymphony.com/quartz/

The *Game Machines* subcomponent uses a number of predefined thresholds in order to update the question and taboo lists. These thresholds are configurable parameters. They can be adjusted at any time. Appendix A explains how to configure these parameters.

5.2.5 Jena Framework

Jena[5] is an open source Java framework for building Semantic Web applications. It provides an easy API that can be integrated within any Java application. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine.

The *Ontology Representation Generator Component* is a Java application that is integrated with Jena in order to produce ontology OWL files. There is no specific reason to go with OWL specification. RDF specification will work fine as well in this project because the ontology has only a class hierarchy with sub-class and super-class relationships. However, future work may include other relationships that are support by OWL only.

5.2.6 Spring Framework

Spring[6] is an open source layered Java application framework. The main advantages of using Spring framework, are (i) abstraction of the data access code, (ii) providing a rich hierarchy of data access exceptions, independent of any particular persistence product, (iii) facilitating the extraction of configuration values from Java code into XML or properties files enabling the modification of the data source configuration without affecting the code. Spring has many other advantages as well.

The game server and all programs that are deployed in it do not have a direct access to the database. Spring is used to implement the service layer between the game server and database resource. The advantage of using the service layer in general is code decoupling. The database technology can be changed at any time without affecting the code. Moreover, Spring has been used to facilitate the integration of the Quartz Scheduler.

5.2.7 Database

The game server needs to retrieve the question and taboo lists to start a new game session. It also needs a repository to store the knowledge collected from the isolated game sessions. Furthermore, the *Game Mechanics* subcomponent needs to retrieve and update the question and taboo lists. In addition, the O*ntology Builder Component* needs to retrieve the stored graphs from a repository to build the ontologies. In order to provide a centralise repository, MySQL[7] has been installed along with the other software in the *Ontology Refinement System.*

[5]http://jena.sourceforge.net/
[6]http://springframework.org/
[7]http://www.mysql.com/

MySQL is an open source light weight database that provides a command line interface with simple commands to program. It has been used as a repository to retrieve and store data required by all the components in the *Ontology Refinement System.*

5.2.8 Eclipse

Eclipse [8] is an open source integrated development environment (IDE). It provides many tools that can be used during the development cycle of a program. It facilitates the design, development, debugging, and deployment of an application. It mainly supports Java application development. However, it can be used for other programming languages as well. It can be easily integrated with other tools like servers.

Eclipse has been used as an editor during the development of the *Ontology Refinement System's* components. It has been integrated with Apache Tomcat to facilitate the deployment of the application.

5.3 Game Interface

This section explores the Graphical User Interface, GUI, of the game which is basically the implementation of the *Game Basics* subcomponent. It provides screen shots of the different states of the game. Moreover, it explains the various features of the game.

When a player accesses the game through the game URL, he or she will be presented with the login screen which is illustrated in figure 5.2. The player needs to provide a player ID and press on the login button in order to play the game.

If the player ID that the player provided is identical to another player's ID who has already logged into the game, the player will receive a message stating that he or she needs to choose another ID and login again like in figure 5.3. The player needs to access the game again providing a different player ID.

If the player provides a unique player ID which is not used by any other logged in player, then the player will login successfully and he or she will be presented with a message like in figure 5.4.

After logging successfully, the server will try to find another logged in player who is not engaged in any game session. If there are no free players, then the server will prompt the player with a message stating that it is trying to find a partner. If it is the case, then the player needs to wait for some time in order to play the game.

Whenever the server finds a free player, the two players will be matched together and a new game session will start. The server matches the players randomly, thus the chances of cheating is eliminated. When the game starts, the players will be presented with the game screen which looks like figure 5.5. Both of the players will see the exact screen.

[8]http://www.eclipse.org/

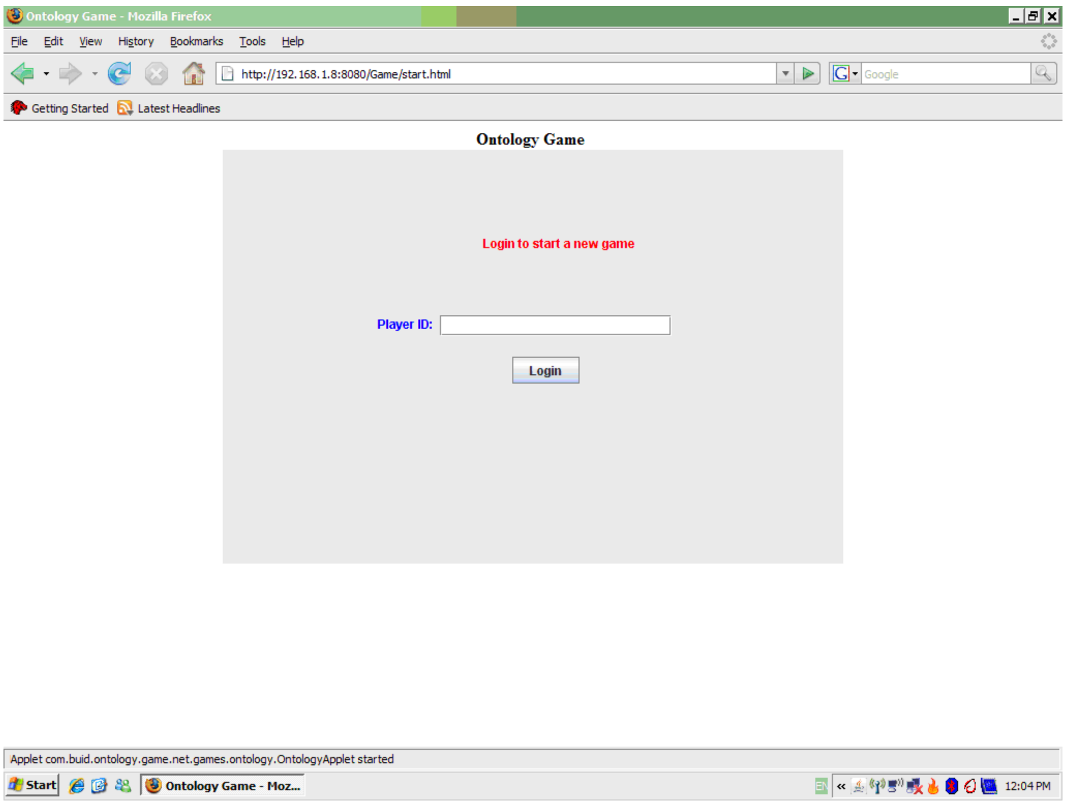

Ontology Game - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.1.8:8080/Game/start.html
Getting Started Latest Headlines
Ontology Game
Login to start a new game
Player ID:
Login
Applet com.buid.ontology.game.net.games.ontology.OntologyApplet started
Start
Ontology Game - Moz...
12:04 PM

Figure 5.2: Login Screen

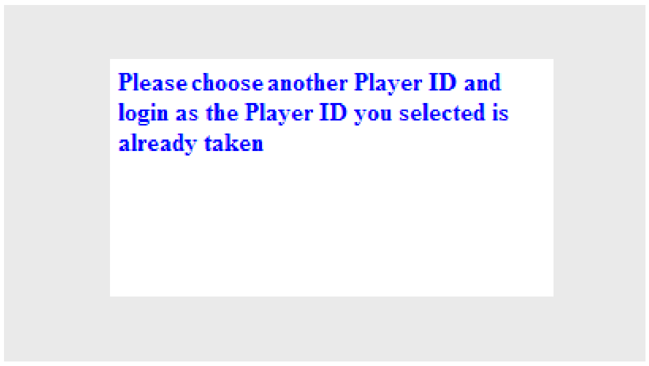
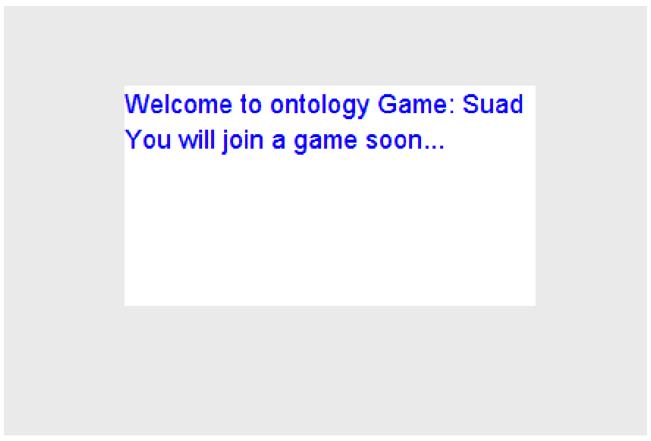

Please choose another Player ID and login as the Player ID you selected is already taken

Figure 5.3: Failure Login Screen

The main game screen has a number of components as illustrated in figure 5.5. These components are as follow:

1. **Topic:** This represents the context of the question being asked. It has been added to overcome the problem of ontologies overlapping. Different ontologies might have the same concepts which are interpreted differently. This can cause confusion for the players. Before reading the question, the player should have a look at the topic to decrease the chances of misunderstanding and increase the chances of earning points.

2. **Question:** This is the question the player needs to answer. It includes the concept where in which sub-classes need to be found.



Welcome to ontology Game: Suad
You will join a game soon...

Figure 5.4: Successful Login Screen

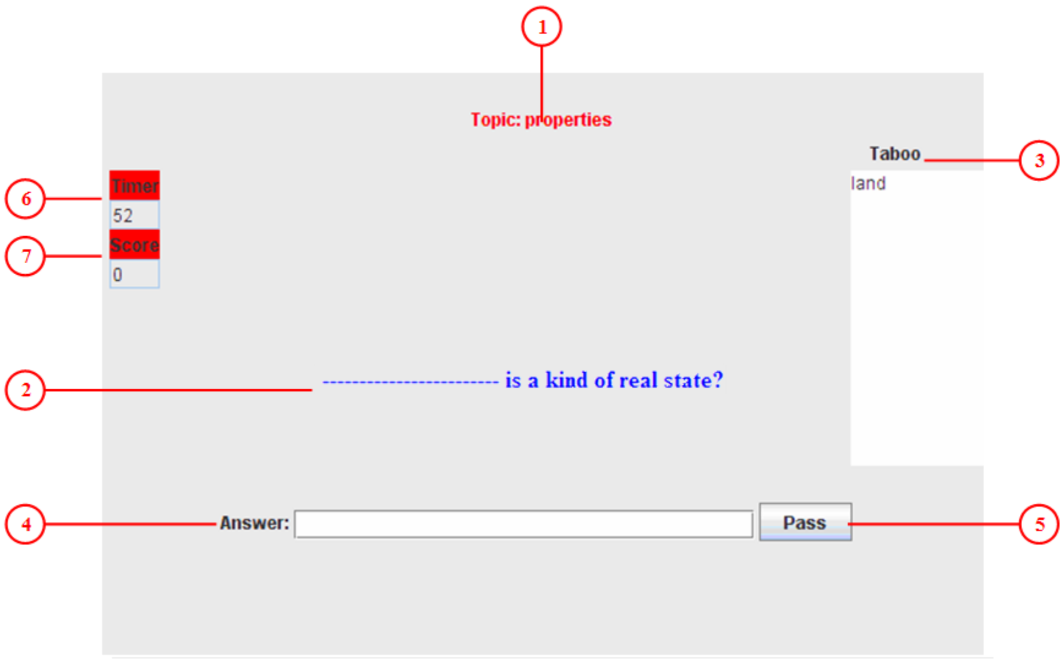


Figure 5.5: The Main Game Screen

3. **Taboo:** This represents the list of words that the player is not allowed to provide as an answer. It actually represents the sub-classes that have been already identified by other players in previous game sessions.

4. **Answer:** The player should type his or her answer inside the box and then hit enter. If the player's answer matches his or her partner's answer, then the question and its taboo list will change automatically. The player's answer should not be included in the taboo list. Moreover, it should not match the concept in the question. The answer should contain more than one character. These are basic rules that have been implemented to validate the player's answer and reduce the chance of cheating. More validations can be added to any extension of the game.

   The player's answer represents a sub-class to the concept in the question. The player can provide as many answers as he or she can think of to a question until an answer matches his or her partner's answer.

5. **Pass:** If the question is hard or the player can not agree with his or her partner on an answer to the question, then the player can request his or her partner to change the question. In order to do this, the player should click on the pass button. The player's partner will receive a message like in figure 5.6. If his or her partner agrees to pass by hitting the pass button as well, then the question and its associated taboo list will change automatically.

6. **Time:** It displays the remaining time of the game in seconds. The time is a configurable parameter that can be changed as explained in appendix A.

7. **Score:** It displays the player's score so far. Both the players in the game will have the same score. The players score increases whenever they agree on an answer.

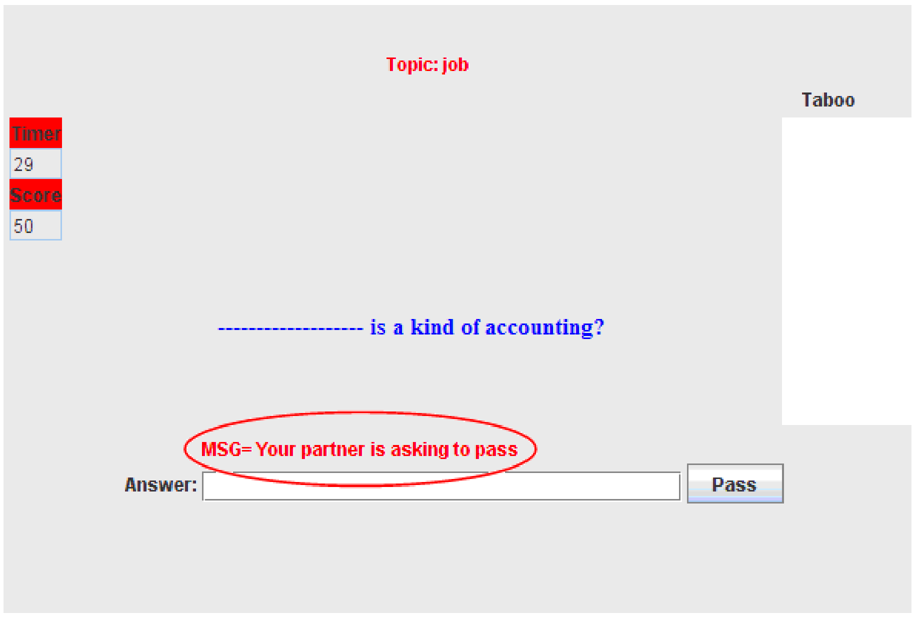

Topic: job
Taboo
Timer
29
Score
50
------------------ is a kind of accounting?
MSG= Your partner is asking to pass
Answer:
Pass

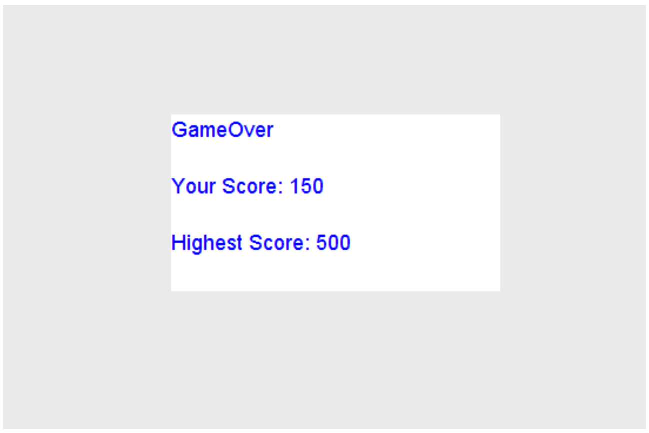
Figure 5.6: Pass Request Screen

When the game is over, the player will be presented with his or her score as will as the highest score in all the game's session. Figure 5.7 illustrates the score's screen.

5.4 Sample OWL File

The aim of the *Ontology Refinement System* is to speed up the process of building ontologies. The *Ontology Representation Generator Component* is responsible for generating the ontology files. It takes a set of statements and outputs an ontology file. The ontology file is written in OWL–DL specification. Figure 5.8 represents a sample OWL file produced by the *Ontology Representation Generator Component*.

The OWL file illustrated in figure 5.8 represents a **"Transportation Ontology"**. It has a unique namespace for all the classes which is a combination of the name of the class and the static URI **http://buid.ac.ae/owl/ontologies/transporta tion/#**.


GameOver

Your Score: 150

Highest Score: 500

Figure 5.7: Gameover Screen

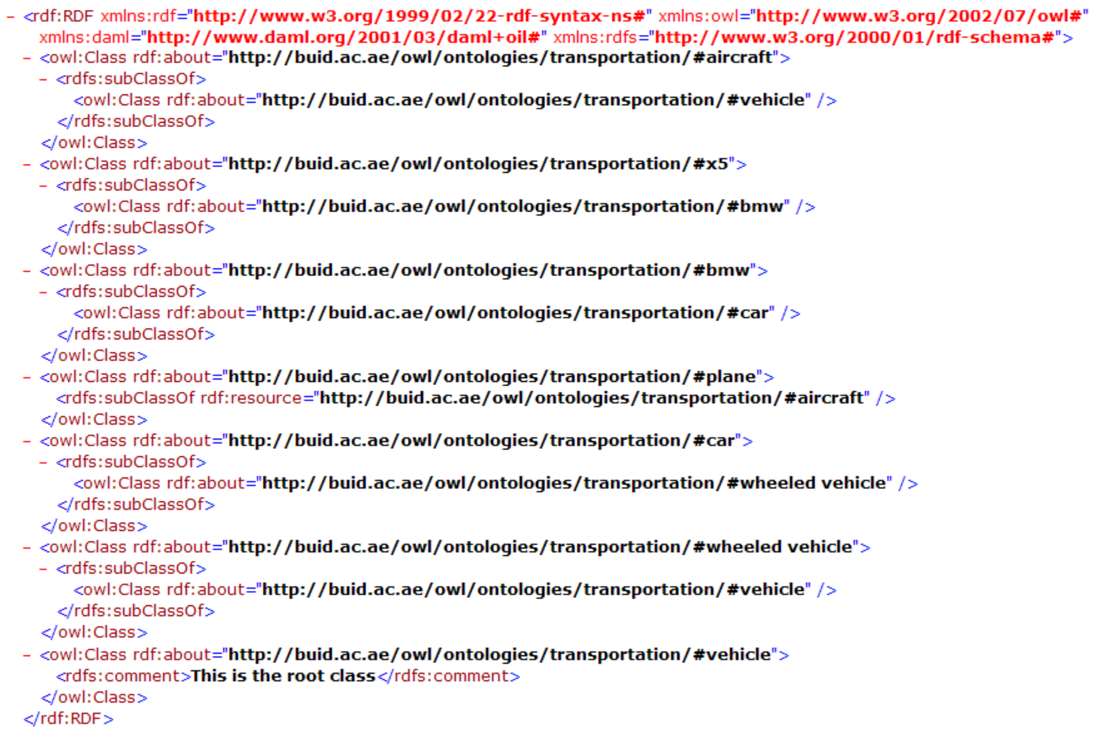
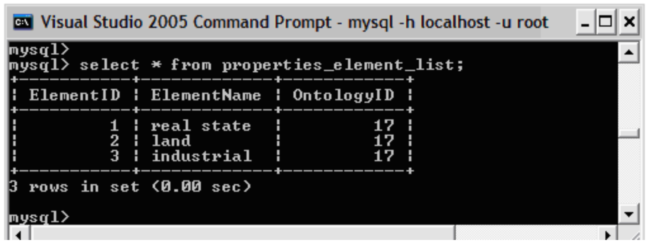

```
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#aircraft">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#vehicle" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#x5">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#bmw" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#bmw">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#car" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#plane">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/transportation/#aircraft" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#car">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#wheeled vehicle" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#wheeled vehicle">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#vehicle" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#vehicle">
      <rdfs:comment>This is the root class</rdfs:comment>
    </owl:Class>
  </rdf:RDF>
```

Figure 5.8: Sample Ontology Output File (OWL Format)

The root class which is **vehicle**, has a comment stating that it is a root class. All the other classes have a "is–sub-class–of" relationship associated with them.

5.5 Internal Representation of the Data Structure

The data structure that has been used in the *Ontology Builder Component* is a graph. Each graph represents an ontology. The graph has been internally stored in two different tables in the database. The first table stores the index and the description of all the unique answers i.e. concepts that the players agreed on at least once. Those concepts are represented later in the graph as vertices. The second table stores the number of times the players have agreed on the fact that there is a "is-sub-class-of" relation between the two concepts. The table uses the concept index from the first table as a reference to these concepts. The number of times the players have agreed that there is a relation between two concepts actually represents the edge's weight. Figure 5.9 and figure 5.10 represent sample screen shots of each table.

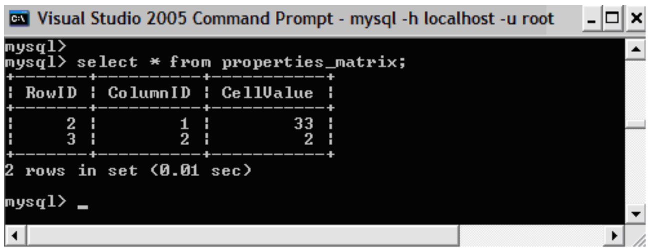


Visual Studio 2005 Command Prompt - mysql -h localhost -u root

```
mysql>
mysql> select * from properties_element_list;
+-----------+-------------+------------+
| ElementID | ElementName | OntologyID |
+-----------+-------------+------------+
|         1 | real state  |         17 |
|         2 | land        |         17 |
|         3 | industrial  |         17 |
+-----------+-------------+------------+
3 rows in set (0.00 sec)

mysql>
```

Figure 5.9: Concept Table



Visual Studio 2005 Command Prompt - mysql -h localhost -u root

```
mysql>
mysql> select * from properties_matrix;
+-------+----------+-----------+
| RowID | ColumnID | CellValue |
+-------+----------+-----------+
|     2 |        1 |        33 |
|     3 |        2 |         2 |
+-------+----------+-----------+
2 rows in set (0.01 sec)

mysql> _
```

Figure 5.10: Weight Table

The screen shots represent the tables of the **"Real State Ontology"**. As you can notice from the first table screen shot, the answers that the players agreed on are **"Land"** and **"Industrial"**. The number of times the players have agreed that **"Land"** is a sub-class of **"Real State"** is 33.

The graph has been represented in two different tables to facilitate the transformation of a graph to a proper data structure in Java. In Java, the second table was used to transform the internal representation of the graph into a matrix using Java array. Array provides easy navigation through the elements in a matrix with a high performance.

The *Ontology Builder Component* outputs a set of statements. These statements contain the indices of the elements. The first table is used to map the indices of the elements in the statements to their description before sending the statements to the *Ontology Representation Generator Component.* This assures that the statements that are used to produce the ontology file contain the actual class description.

5.6 Technical Limitations

The game suffers from four technical limitations that can be fixed if more time were provided. The limitations do not have any major impact on the functionalities of the game as a prototype. However, the game can not be moved to production with these limitations as it is hard to manage with a huge number of players.

The first limitation of the game is that the applet can not be initiated in Internet Explorer. It can be initiated using Mozilla FireFox. Other browsers have not been tested. The reason for that has not been investigated yet.

The second technical limitation is that the game does not have proper registration for the players. The player ID is used to uniquely identify players. Therefore if two different players are using the same ID, the second player will receive a message stating that he or she needs to access the game again with a different player ID which is sometimes annoying. This can be solved through the development of proper registration module that stores registered players unique IDs.

The third technical limitation is that a player can only log in once from a machine. If the player tries to log in more than once from the same machine with another player ID, both of the game sessions will receive the same events from the server as the server sends events to the client based on the client IP and socket. This can be solved by either blocking the player from logging in more than once or having some unique ID per game session that the client can use to filter the events received from the server and process only the events that belong to the particular game session.

The last and major limitation is refreshing or closing the browse issue. If any player refreshes or closes his or her browser before the game gets over, the player will never be logged out from the system i.e. game server. Therefore, any player who will be matched with this player, will never get any response from his or her partner. In order to solve the problem, a scheduled job should be implemented that clears out i.e. logs out all the inactive players after a certain period of time.

Chapter 6

Experiment Design

In order to evaluate the usability and efficiency of the *Ontology Refinement System*, an experiment has been conducted. The experiment focuses on evaluating the game interface which is the *Game Basics* subcomponent and the algorithm design which is represented by the *Ontology Builder Component.*

This chapter explores the design of the experiment. It explains the initial data that has been used to run the game. It briefly introduces the manual that has been prepared in order to ease the process of running the experiment anywhere. Moreover, it discuses the parameters that can be adjusted based on the volume of the experiment i.e. the number of volunteers and time given to play the game. It also describes the environment where the experiment has taken place. Finally, it describes the surveys that have been used to provide feedback about the game and to evaluate the ontologies produced by the *Ontology Refinement System.*

6.1 Initial Data

The game requires a small set of predefined question and taboo lists in order to run the first few sessions. Later game sessions will use the initial data as well as the players' answers. The predefined question and taboo lists should belong to restricted domains i.e. ontologies to increase the opportunity of producing intensive ontologies i.e. ontologies that contain many concepts.

The domains that have been selected for the experiments are "Animal", "Transportation" and "Food Classification". These domains cover different areas of interest and commonsense knowledge. Intensive ontologies that represent these domains have been defined prior to the experiment. These ontologies are used to build the question and taboo lists of the experiment. In addition, they can be used to evaluate the ontologies produced by the experiment.

The "Animal" ontology[1] is an existing ontology that contains 417 concepts. The

[1]http://www.cs.man.ac.uk/r̃ector/modules/CS646/Lab-Material/animal-onts-lect-2/Animals-tutorial-step-0.owl

"Transportation" ontology[2] is a mix of two existing ontologies and the author defined ontology. The author's concepts have been introduced because the existing ontologies are small. The total number of concepts in both of the existing ontologies is 117. However, the relations between these concepts are mostly not a "is-sub-class-of" which is not the main focus of the *Ontology Refinement System*. Only 18 concepts have "is-sub-class-of" relation associated with them. The "Food Classification" ontology is purely defined by the author. Existing ontologies have been used for reusability and ease of evaluating the ontologies produced by the experiment.

A subset of the concepts defined in the three ontologies has been used to construct the question and taboo lists i.e. the initial data of the experiment. Table 6.1 summarised the initial data.

Ontology	**Question**	**Taboo List**
Animal	________ is a kind of animal?	Mammal Reptile
	________ is a kind of mammal?	Cat
	________ is a kind of reptile?	Crocodile
	________ is a kind of crocodile?	
	________ is a kind of cat?	Domestic cat
	________ is a kind of domestic cat?	
Transportation	________ is a kind of vehicle?	Wheeled vehicle Aircraft
	________ is a kind of wheeled vehicle?	
	________ is a kind of aircraft?	
Food Classification	________ is a kind of food classification?	Proteins
	________ is a kind of proteins?	Dairy product
	________ is a kind of dairy product?	Cheese
	________ is a kind of cheese?	

Table 6.1: Experiment Initial Data

[2]http://www.xml.com/lpt/a/1531
http://faure.isti.cnr.it/ straccia/download/teaching/autos.owl

6.2 Experiment Manual

The *Ontology Refinement System* should be installed in a machine that can be accessed through the Internet or a network. Moreover, it requires a number of software to be installed along with it in the server and in the client i.e. players' machines as well. In order to facilitate running the experiment anywhere and at anytime, an experiment manual has been prepared. It provides detailed guidelines of how to run the experiment. The instructions that the manual provides are for the administrators who want to conduct the experiment as well as the volunteers who will be playing the game.

The experiment manual explains all the prerequisites needed to install the *Ontology Refinement System*. The manual is divided into two sections. These sections are the administrator section and the volunteers section. The administrator section provides installation, testing, and troubleshooting guidelines of the game server, Web server and game's database. It explains the steps that the administrator needs to take into account while running the experiment such as providing the guidelines to the volunteers prior to playing the game. It describes the process of extracting the output after conducting the experiment. It also explains the configurable parameters and how to adjust them. Finally, it provides a copy of the survey that should be filled by the volunteers.

The volunteers section introduces the game. It explains in details the components in the main screen of the game. It provides the list of prerequisites that the volunteers need to have in order to play the game. Moreover, it describes the different states of the game supplemented with sample screen shots. It provides the link to the online copy of the survey which the volunteers need to fill after playing the game. The administrator can use the provided copy of the survey in case the volunteers can not access the Website. Finally, the manual discusses the tips that the volunteers need to follow in order to avoid the technical limitations of the game that is explained in section 5.6. For more information about the manual, refer to appendix A.

6.3 Parameters Configuration

The *Ontology Refinement System* defines a number of configurable parameters that can be adjusted at any time. Most of these parameters are used by the *Game Mechanics* subcomponent in order to update the game's database as described in section 4.2.2. The dependencies between these parameters need to be considered when adjusting their values. These dependencies are explained in section 4.2.2 and section 4.3.2. Moreover, the volume of the experiment i.e. the number of volunteers and time given to play the game need to be considered when adjusting these parameters. For huge experiments, the values should be high enough to reduce the noise, while for small experiment the values should be low enough to enable the players to see different questions generated by their answers.

The experiment conducted in this thesis is a small experiment. The total number of players was 20 and the time given to play the game was around 2 hours. Thus, the parameters were set to low values. Table 6.2 provides the values of the parameters set for the experiment.

Name	**Value**
Question Threshold	3
Pass Threshold	30
Element Threshold	100
Taboo List Threshold	10
Taboo Threshold	4
No. of questions to be displayed	10

Table 6.2: Parameters Configuration of the Experiment

The number of times the players would agree on an answer to a question is expected to be low; therefore the *Question Threshold* and the *Taboo Threshold* are set to low values. The number of times the players would agree on passing a question is expected to be medium; therefore, the *Pass Threshold* is set to a high value. This ensures that the question will not be removed fast. The *Element Threshold* is set to a very high value because players might agree once or twice on an answer to a question. Thus the number of various answers to a question will be very high but most of these answers are not eligible to be a question. In addition, the number of elements in the taboo list is not except to be high as the number of times the players might agree on an answer to a question is low. Therefore the *Taboo List Threshold* is set to a low value. The number of questions the players will view in a single game session is expected to be low because for a question to change the players need to agree on an answer or agree to pass the question, and the duration of each game session is only 2 minutes. In case the players have viewed all the questions in a session and the game is not over yet, then the same set of questions will be displayed again to the players.

6.4 Environment

The experiment has been conducted in the university lab which has 25 machines. All the prerequisites software have been installed prior to the experiment. This includes the server machine and volunteers' machines. The total number of volunteers was 20. The volunteers are somewhat of different backgrounds. The volunteers were asked to be present in the lab at a predefined time. This ensures that there are enough players to play the game. The players were asked not to communicate with each other while playing the game to avoid cheating. The author was present during the experiment to monitor the experiment. Before playing the game, the volunteers were introduced to the game using the experiment manual. The experiment has been conducted twice with the same volunteers each session last for 1 hour. After playing the game, the volunteers were asked to provide their feedback about the game by filling the survey.

6.5 Evaluation Surveys

Two different surveys were used to evaluate the *Ontology Refinement System.* The first survey has been conducted by the volunteers who played the game. It was used to collect background knowledge about the volunteers as well as their feedback about the game. It aims to evaluate the game interface and the flow of the game. The volunteers were asked to fill the survey after playing the game. The survey can be found in section A.2.7.

The second survey, "Ontology Evaluation Survey", has been conducted by a random set of Internet users. The survey was used as an evaluation means to evaluate the algorithm implemented in the *Ontology Builder Component.* The volunteers were asked to provide their degree of agreements about the ontologies produced by the *Ontology Builder Component* using the knowledge collected during the experiment. The "Ontology Evaluation Survey" survey can be found in appendix B.

Chapter 7

Evaluation

This chapter discusses and analysis the result of the experiment described in chapter 6. It provides the volunteers feedback about the game. It discusses the knowledge collected during the experiment. Moreover, it presents one of the ontologies produced by the *Ontology Builder* and *Ontology Representation Generator Components* from the knowledge collected during the experiment. Finally, it describes and analysis the result of the "Ontology Evaluation Survey" that has been conducted to evaluate the ontologies produced.

7.1 Experiment's Volunteers

The experiment has been conducted in two different sessions. Each session lasts for an hour. The total number of volunteers who participated in the experiment is 20. In the first session 18 volunteers were present while in the second session 10 volunteers were present. 8 of the volunteers who were present in the second session were also present in the first session. In addition, two new volunteers have joined the second session of the experiment. This section provides the volunteers background and their feedback about the game.

7.1.1 Background

The volunteers have different backgrounds. 11 of the volunteers hold a degree in Computer Science, 5 of the volunteers hold a degree in Engineering, and 2 of the volunteers hold a degree in Business. Furthermore, one of the volunteers holds a degree in Computer Science and Education and another volunteer holds a degree in Computer Science and Business. All the volunteers are proceeding their master degree except one of the volunteers who holds a PhD.

The volunteers are mixed of both genders. 9 of the volunteers are male and 11 of them are female. The volunteers' ages range from below 18 to 34. One volunteer is below 18 years old. The age of 6 volunteers is between 18 and 24. The age of 11 volunteers is between 25 and 30, and the age of 2 volunteers is between 31 and 34.

7.1.2 Feedback

The most important thing to note is that the volunteers have filled the survey after the first session of the experiment. In this session, some of the volunteers joined the experiment late therefore; they did not have enough background about the game. Due to that, they did not follow the tips to avoid the technical limitations of the game which result in players refreshing the browsers. This disturbed the other players. Thus, the feedback gathered was affected by these issues. In the second session of the game, this problem was eliminated. All the volunteers were notified to avoid refreshing their browsers while playing the game. Thus, the knowledge collected from the second session of the experiment was better than the first session.

The volunteers were able to understand the flow of the game easily. In the first session of the experiment, the highest score was 250 which means that there were two players who have agreed on three different answers to different questions in one game session. In the second game session, the highest score was 500. This clearly shows that the more the volunteers were playing the game, the more they understood and enjoyed the game. If the game was played again, the amount of knowledge collected will be at least doubled.

After the first session of the game, the volunteers were asked to rate from 1 to 5 how easy it is to understand the concept of the game. 60% of the volunteers considered the game to be very easy, 25% of the volunteers considered the game to be somewhat easy, 10% of the volunteers were undecided, and only 5% of the volunteers considered the game to be somewhat difficult. These volunteers, who considered the game to be somewhat difficult, declared in their comments that the game is not understandable. Despite the fact that the game was not explained to all of the volunteers, only 15% of the volunteers did not consider the game to be easy which clearly states that the game is well defined.

The volunteers were also asked to state what did they like the most about the game. The following list provides quotes from the volunteers' answers which show that the game is as enjoyable as the other "Game with a Purpose" series that Luis [16] developed.

- Being Random.
- The concept. The questions that change each time which makes you so excited to know what's next.
- I found that the idea of the game is new and it may help in building ontologies.
- That the other player was unknown.
- Guessing words and increasing vocabulary.
- Innovative concept and ease of learning.
- The idea.
- Taboo.

The volunteers were also asked to state what they did not like about the game. The majority of the answers were related to the game interface which is planed to be enhanced in future work. 20% of the volunteers were either confused about clicking the pass button to provide an answer instead of hitting enter or annoyed by too many pass requests from their partners which was because of the former. One of the volunteers suggested to place the pass button somewhere else in the screen or to provide another button to enter the answer. The author believes that hitting enter to provide an answer makes it faster to provide many answers in a short period of time as the player does not need to move his or her hand away from the keyboard. Placing the pass button next the area where the answers should be provided was meant to speed up the process of sending a pass request and providing answers to the questions. The same design is followed in all the games developed by Luis [18, 17, 19, 20]. It just needs time to get used to it. 10% of the volunteers stated that they did not like the interface of the game. 15% of the volunteers complained about the technical problems caused by refreshing the Internet browsers by the other volunteers. This problem has been controlled in the second session of the experiment. 5% of the volunteers stated that the questions were difficult and another 5% of the volunteers stated that it is difficult to agree on an answer with their partners. This is expected in the first few sessions of any multiplayer game. One of the volunteers suggested displaying the answers that matches at the end of each game, which is a good approach to show the kind of answers that are expected. Another volunteer suggested having more options when the game is over such as replay which is another good suggestion. The game lacks many facilities because it is designed to be used as a prototype to support the thesis statements.

Moreover, the volunteers were asked whether they would like to play the game. 55% of the volunteers were welling to play the game again while 45% of the volunteers were not welling to play the game again. 44% of the volunteers who were not welling to play the game again have stated that they were either annoyed by the technical problems or by the too many pass requests. So, if we do not consider these volunteers as these two issued were resolved in the second session of the game, then around 70% of the volunteers would have been willing to play the game again.

Furthermore, the volunteers were asked to rate from 1 to 5 how interesting the game is. 15% of the volunteers stated that the game is very interesting, 50% of the volunteers stated that the game is somewhat interesting, 20% of the volunteers were undecided, 10% of the volunteers stated that the game is somewhat boring, and 5% of the volunteers stated that the game is very boring. 66.7% of the volunteers who stated that the game is boring were again either annoyed by the technical problems or too many pass requests. So, if we do not consider these volunteers, then 72% of the volunteers would have stated that the game is interesting.

The overall of the game feedback was good. The difficulties that the volunteers were facing were due to the lack of understanding and not following the guidelines which led to some technical issues to show up. These issues were annoying to the volunteers. The game can be enhanced by solving these technical problems and providing proper guidelines to all the volunteers.

7.2 Gathered Knowledge

The total number of game sessions that the volunteers played in the two experiment sessions is 57. The game sessions in the first experiment session is 23, while the game sessions in the second experiment session is 34. The total number of agreement between the players on different answers is 56. 21 agreements were in the first experiment session and 35 agreements were in the second experiment session. Although, the number of volunteers in the second experiment session is almost half of the number of volunteers in the first experiment session, the number of agreements has increased up to 25% in the second experiment session which is another indication that the volunteers understood and enjoyed the game more in the second session.

One observation to state about the players' answers is that in the first experiment session 39% of the game sessions were ended with at least one agreement on an answer, while in the second game session the ratio has increased to around 60%. This is another evidence that with clear explanation of the game and with avoiding the game limitations, the success of the game increases.

Table 7.1 represents the various answers provided by the volunteers and the total number of agreements on these answers for the "Transportation" ontology. The answers provided for the "Food Classification" ontology and "Animal" ontology can be found in section C.1. Note that the sub–class column represents the answers, while the super–class column represents the concepts in the questions.

An inserting observation of the gathered knowledge is that the players were moving the answers from general to specific. For example, Plane was stated as a sub–class of Vehicle. It was also stated that it is a sub–class of Aircraft which is more specialised. Another observation is that the total number of agreements is low. This is expected as the volume of the experiment is small.

7.3 Ontologies Produced

The knowledge gathered from the experiment was stored in a database. The *Ontology Builder Component* was used to retrieve this knowledge and produce ontologies based on this knowledge. The *Ontology Builder Component* defines two parameters as explained in section 4.3.2. The maximum values that can be set to these parameters are 3 for the <u>*Node Threshold*</u> and 2 for the <u>*Parent Threshold*</u>. This is due to the fact that the <u>*Node Threshold*</u> should be equal to or less than the <u>*Question Threshold*</u> which is set to 3 and the <u>*Parent Threshold*</u> should be less than the <u>*Node Threshold*</u> .

In this case, the algorithm will only consider 12 associations from the tables 7.1, C.1, and C.2. These associations are the ones were the total number of agreement is greater than 3. The *Ontology Builder Component* produces three ontologies. The break down of the concepts in each ontology is explained in table 7.2.

If the values of the thresholds are set to even lower values such as the <u>*Node Threshold*</u>

Sub–class	**Super–class**	**Agreements count**
Plane	Aircraft	3
Car	Vehicle	8
Car	Wheeled Vehicle	6
BMW	Car	9
Nissan	Car	1
Honda	Car	1
Toyota	Car	2
Plane	Vehicle	2
BMW	Vehicle	1
Four wheel	Vehicle	2
X5	BMW	3
Nissan	Vehicle	1
Bus	Vehicle	1
Truck	Vehicle	2
4.6	X5	1

Table 7.1: Experiment Knowledge Collected for the Transportation Ontology

is sent to 2 and the *Parent Threshold* is set to 1, then the size of the ontologies will change as shown in table 7.3.

The *Ontology Builder Component* produces the ontologies as a set of statements and then sends them to the *Ontology Representation Generator Component* which in turns generates an OWL file representing each ontology. Figure 7.1 represents the OWL file generated for the "Transportation" ontology. The OWL files generated for the "Animal" ontology and "Food Classification" ontology can be found in section C.2. These OWL files represent the ontologies produced when setting the *Node Threshold* to 2 and *Parent Threshold* to 1.

The hierarchical relationship between the concepts in the "Transportation" ontology are shown in figure 7.2. The concepts that are added by the volunteers are represented in red boxes.

Ontology Name	**Predefined Concepts by the author**	**Concepts Introduced by the players**	**Total No. Of Concepts**
Animal	6	1 (12 ignored)	7
Food Classification	4	3 (4 ignored)	7
Transportation	3	4 (8 ignored)	7

Table 7.2: Break Down of the Concepts in the Ontologies Produced (1)

Ontology Name	**Predefined Concepts by the author**	**Concepts Introduced by the players**	**Total No. Of Concepts**
Animal	6	6 (7 ignored)	12
Food Classification	4	4 (3 ignored)	8
Transportation	3	8 (4 ignored)	11

Table 7.3: Break Down of the Concepts in the Ontologies Produced (2)

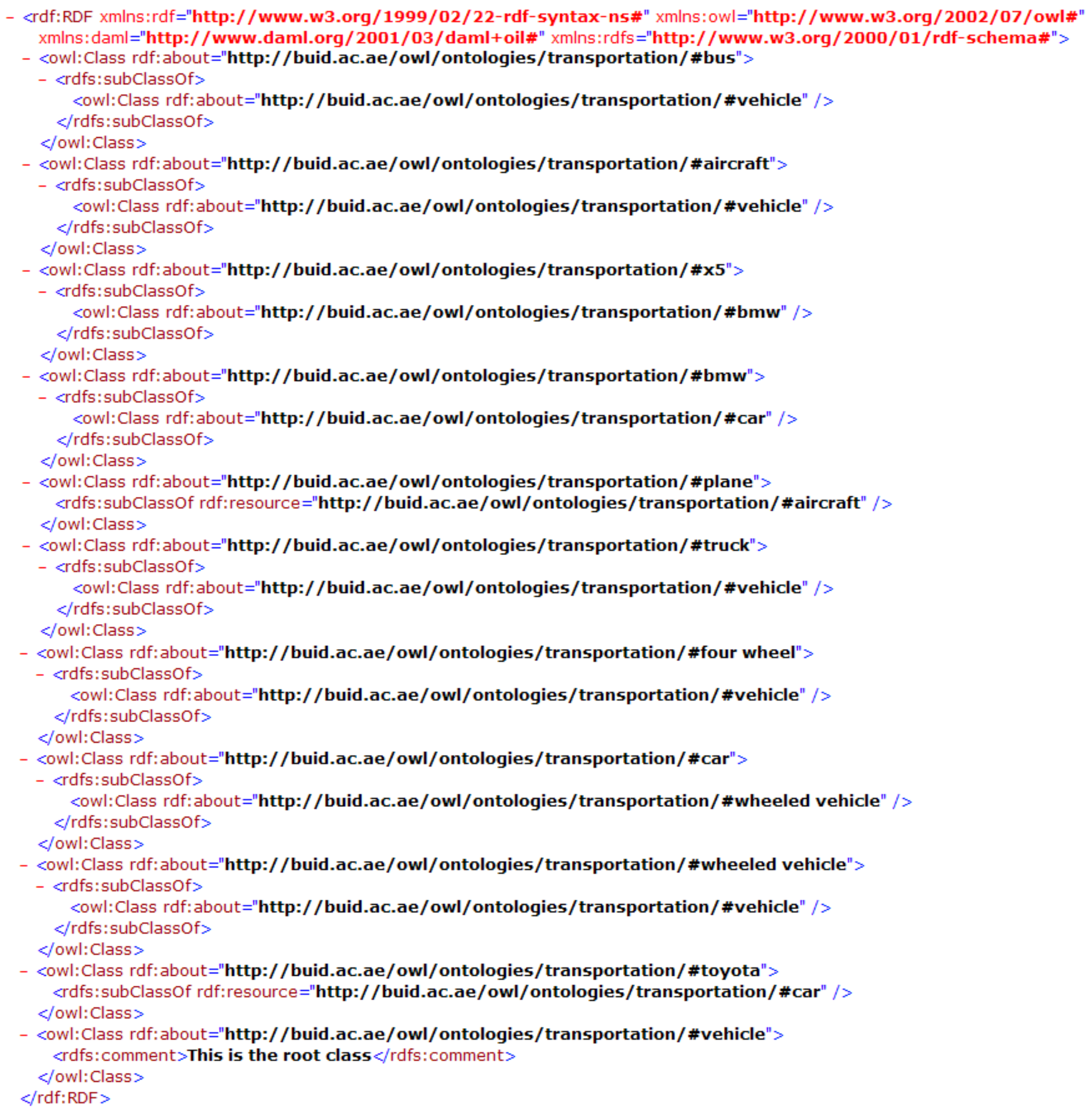
The algorithm produces the minimal associations. For instance, in the case of the Car concept, Car is a Vehicle and Car is a Wheeled Vehicle. The algorithm infers that Car is a Wheeled Vehicle because Wheeled Vehicle is a Vehicle. So, Car is also a Vehicle which is implicitly known since it follows from the first two and from the transitivity of the “sub–class” relation. Thus, there is no need to present it explicitly.

7.4 Ontology Evaluation

The ontologies produced by the *Ontology Refinement System* need to be evaluated in terms of its accuracy and efficiency. In order to do so, any of the evaluation means discussed in section 2.1.2 can be used. Any automated tool developed to measure the alignment between the ontologies can be used to compare the ontologies produced and the predefined ontologies. However, because the size of the ontologies produced is small in terms of the number of concepts they represent, a survey has been conducted to evaluate the ontologies.

The survey aims to find the degree of people agreements on the statements that have been generated by the *Ontology Builder Component*, specially the statements that have been inferred from the knowledge collected by the game and not the initial data of the game. The survey is provided in appendix B.

The survey has been distributed randomly to Internet users. The number of volunteers who have filled the survey is 140. These volunteers have diverse backgrounds. 32% of the volunteers hold a degree in Computer Science, 15.7% of the volunteers hold a degree in Business Science, 14.3% of the volunteers hold a degree in Engineering, 10.7%



```
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#bus">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#vehicle" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#aircraft">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#vehicle" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#x5">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#bmw" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#bmw">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#car" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#plane">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/transportation/#aircraft" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#truck">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#vehicle" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#four wheel">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#vehicle" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#car">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#wheeled vehicle" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#wheeled vehicle">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#vehicle" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#toyota">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/transportation/#car" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/transportation/#vehicle">
      <rdfs:comment>This is the root class</rdfs:comment>
    </owl:Class>
  </rdf:RDF>
```

Figure 7.1: Transportation Ontology in OWL Format

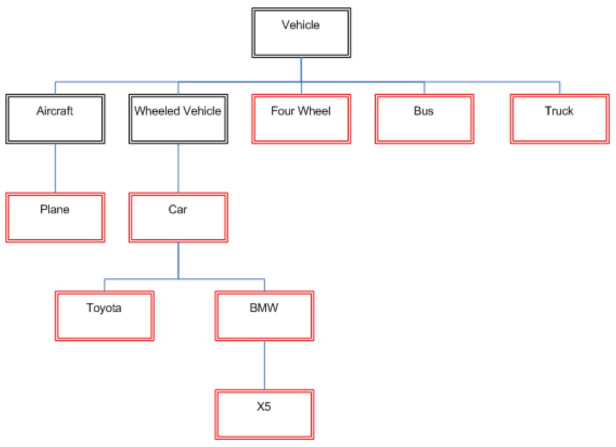


Figure 7.2: Transportation Ontology hierarchy

of the volunteers hold a degree in Education, 10.7% of the volunteers hold a degree in Art and Science, 2.3% of the volunteers hold a degree in Medicine, and 14.3% of the volunteers are still in high school. Moreover, the level of education is various among the volunteers. It includes high school, Diploma, Higher Diploma, Bachelor, Master, and PhD.

The volunteers are from a different age span. 16.4% of the volunteers are below 20 years old, 34.3% of the volunteers are between 20 and 25 years old, 30% of the volunteers are between 26 and 30 years old, 10% of the volunteers are between 31 and 35 years old, 3.6% of the volunteers are between 36 and 40 years old, and 5.7% of the volunteers are above 41 years old. In addition, the volunteers are from both genders. 55.7% of the volunteers are female while 44.3% of the volunteers are male. The volunteers have been selected from different backgrounds, education levels, ages, and gender to simulate the whole population that might be involved in the process of building ontologies therefore, their opinions need to be considered.

The volunteers have been asked to rate from 1 to 5 their level of agreement on the given statements. Figure 7.3 represents the overall agreements on all the statements. Notice that the biggest portion of the chart represents the volunteers who have totally agreed on all the statements. The total percentage of the volunteers who agreed on all the statements is 90.54%. This percentage includes volunteers who totally agreed, somewhat agreed and agreed on the statements. Thus, the overall agreement is excellent. Table 7.4 presents the summarised percentage of the volunteers agreement on each statement. Figure 7.4 visualises the summarised table.

There are many factors that might affect the voulneers agreement such as the understanding of the vocabulary and the culture. For example, people in UAE mostly interpret the term meat to be lamp. This might be the reason for which only 53.6% of the volunteers have totally agreed that fish is meat.

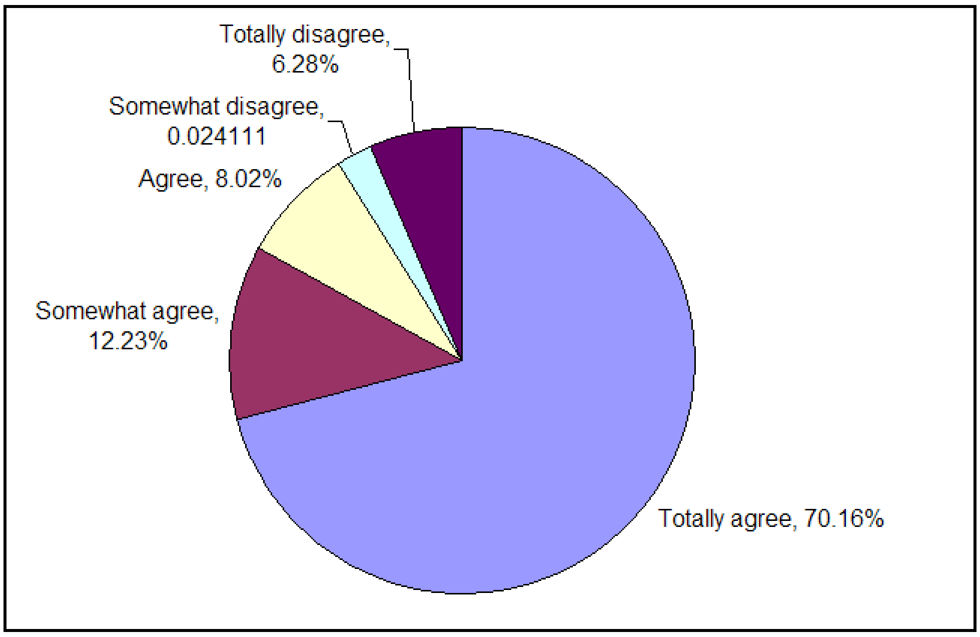


Figure 7.3: Overall Ontology Evaluation Chart

Statements	**Totally agree (1)**	**Somewhat agree (2)**	**Agree (3)**	**Somewhat disagree (4)**	**Totally dis-agree (5)**
Plane is a kind of Aircraft	72.1%	16.4%	4.3%	0	7.1%
Butter is a kind of Dairy Product	59.3%	11%	15.7%	5%	8.6%
Lizard is a kind of Reptile	72.9%	12.9%	10.7%	0	3.6%
BMW is a kind of Car	78.6%	7.9%	5.7%	2.1%	5.7%
Dog is a kind of Mammal	67.9%	13.6%	11%	2.1%	5%
Four Wheel is a kind of Vehicle	67.1%	10.7%	10%	3.6%	8.6%
Fish is a kind of Meat	53.6%	21.4%	11%	3.6%	10%
Bus is a kind of Vehicle	74.3%	12.9%	8.6%	1.4%	2.9%
Monkey is a kind of Mammal	72.9%	12.9%	4.6%	4.3%	3.6%
X5 is a kind of BMW	71.4%	10.7%	10%	2.9%	5%
Snake is a kind of Reptile	76.4%	15%	5%	2.1%	1.4%
Meat is a kind of Proteins	60%	12.1%	10.7%	3.6%	13.6%
Truck is a kind of Vehicle	77.1%	11%	4.6%	1.4%	3.6%
Camel is a kind of Mammal	75%	10%	7.9%	2.9%	4.3%
Toyota is a kind of Car	71.4%	8.6%	8.6%	2.1%	9.3%
Milk is a kind of Dairy Product	69.3%	10%	2.1%	2.1%	7.9%
Cow is a kind of Mammal	72.9%	11%	4.6%	2.1%	7.1%
Car is a kind of Wheeled Vehicle	70.7%	12.1%	9.3%	2.1%	5.7%
Total Average	**70.16%**	**12.23%**	**8.02%**	**0.024%**	**6.28%**

Table 7.4: Ontology Evaluation Survey Result

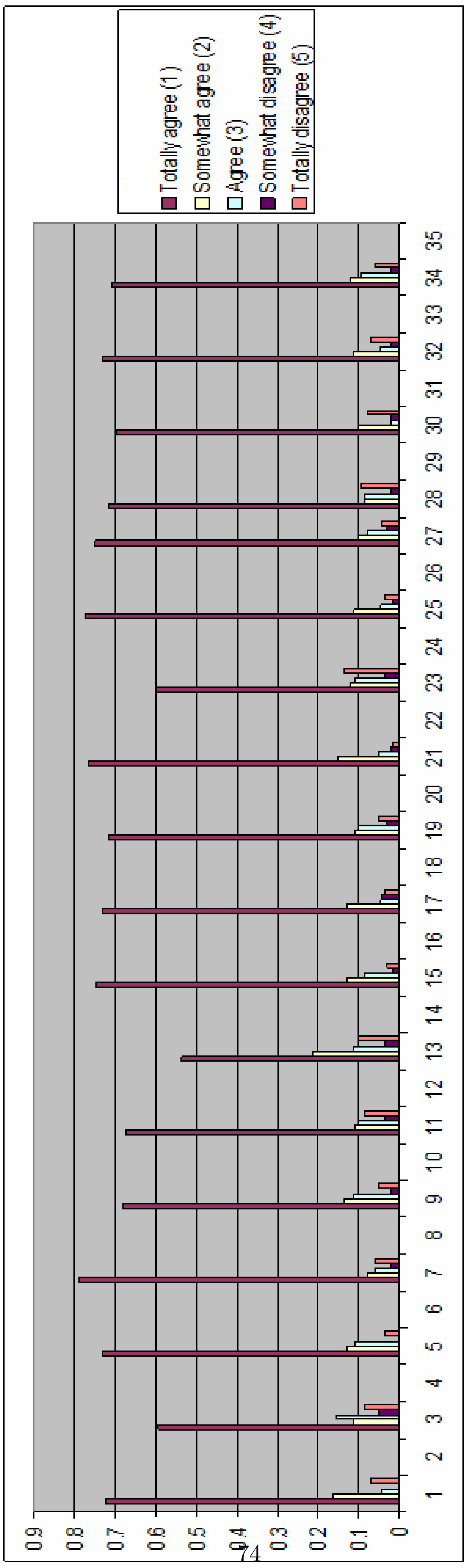


Figure 7.4: Detailed Ontology Evaluation Chart

Chapter 8

Future Work and Conclusion

Ontology is a fundamental backbone for the Semantic Web. It enables organising the information in a structured form which in turns facilitates searching and navigating information on the Web. However, the number of ontologies that are currently available is very minimal. Most of these ontologies are outdated. Moreover, the process of developing ontologies is restricted to experts only.

The *Ontology Refinement System* that has been developed in this thesis aims to speed up the process of building ontologies through an online game. The online game collects commonsense knowledge from the players. People can play the game in their leisure time. The game enables a large community of people to be involved in the process of building up–to–date ontologies. The *Ontology Refinement System* does not only collect commonsense knowledge but also processes this knowledge and transforms it to meaningful ontologies. All the projects that have been developed so far to collect commonsense knowledge lack the aspect of processing the knowledge collected.

The *Ontology Refinement System* handles only the hierarchical relationships of an ontology. Future extension of the system can easily include the other types relationships and all the other aspects of an ontology such as properties and data constrains. Moreover, a text processing tool can be integrated with the *Ontology Refinement System* to identify the root of a word and match the players answers based on the root of the answer and not the identical match. This will enable building precise ontologies as the different variations of the word will not be considered as different concepts i.e. cat and cats. In addition, the game interface should change to a fancier interface in order to attract more players to play the game. A proper registration mechanism needs to be added to the game which allows players to register once and then use the same player ID all the time. Thus, they do not get annoyed by changing the player ID every time if it is in use. Furthermore, the single player mode should be developed to reduce the time the player needs to wait for an opponent. More validation rules can be added to deduct any cheating attempts. Finally, all the technical issues that have been highlighted in the thesis such as refreshing the browser and logging in more than once need to be resolved.

Appendix A

Ontology Game Manual

A.1 Introduction

The Ontology Game is a multiplayer online game. The game logic is inspired by the ESP game. It is a Web–based game for two players. Each player will be matched with an anonymous partner i.e. another online player. The two players will be presented with concepts, and will need to write down all sub–categories they can think of. A player cannot see what his or her partner is typing, but if both write the same sub–category, they will earn points. The player will have the option to pass if the concept presented is hard. The game will last for two minutes only in order to keep it more interesting.

A.1.1 Goals of the Game

The Ontology Game has been developed in order to speed up the process of collecting commonsense knowledge. This knowledge is then processed and represented as ontologies.

A.1.2 Whom the manual is for

The manual is for administrators and professors who want to run the Ontology Game as an experiment on a group of volunteers.

A.1.3 What the Manual Covers

This manual covers all the required steps needed to install and run the game by the administrator. Moreover, it contains a survey that should be given to the volunteers after playing the game. It also includes guidelines for the volunteers, explaining how to play the game.

A.1.4 Manual Supplementary Resources

Along with the manual there is a Prototype.rar file that contains all the required documents, and scripts to install and run the game. Whenever, there is a reference in the manual to the Prototype directory please refer to the extracted directory of this file.

A.1.5 Game Required Resources

In order to install and play the game, there are a number of prerequisites. Following are the list of prerequisites for installing the game and the list of prerequisites for playing the game.

Installing the Game Prerequisites:

1. Sun jre 1.5 or later
2. MySQL 5.0.25 or later
3. Apache Tomcat 5.5 or later
4. A machine with an accessible IP. The volunteers need to connect to this machine to play the game.

Playing the Game Prerequisites:

1. Sun jre 1.5 or later
2. Mozilla FireFox

Author Contact:
Email: suad.alshamsi@gmail.com

A.2 For the Administrator

A.2.1 Installing the Game

Prerequisites:

In order to run the game, install the following software:

1. Sun jre 1.5 or later
2. MySQL 5.0.25 or later
3. Apache Tomcat 5.5 or later

The machine where the game is installed should have an accessible IP. The volunteers need to connect to this machine to play the game.

Database Configuration:

After installing the MySQL database do the following [1] :

1. Start MySQL server from the command line/shell using the following command **mysqld**

   Note that the control does not come back and its an expected behaviour

2. In another command line/shell, import the database schema using the following command
   mysql -h host -u username -p password < mySql.sql

   or if you do not have a password

   mysql -h host -u username < mySql.sql

   ****mySql.sql** can be found in Prototype/Database/mySql.sql directory.
   ****host** is the host name where the database is installed e.g. localhost.
   ****username** is the username default root.
   ****password** is the password default empty.

3. Ensure that the schema has been created successfully by running the following commands **mysql -h host -u username -p password**

   or if you do not have a password

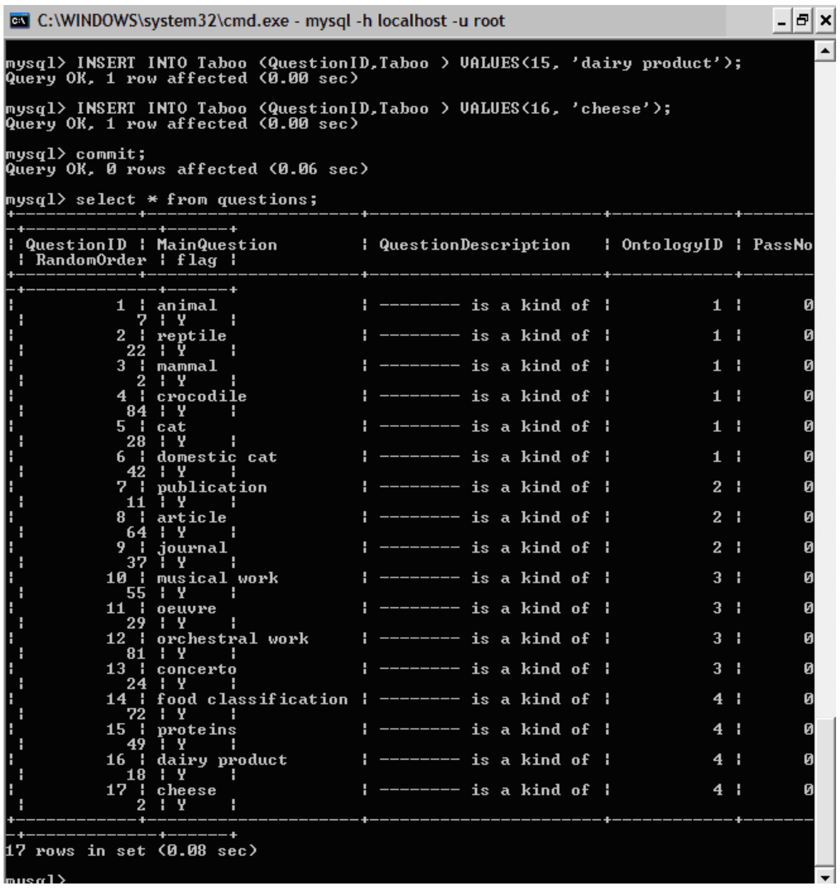
   mysql -h host -u username

[1] Note that the command is given for Windows. The same command will run in any other operating system. If you are facing any problem, then please check mysql reference document.

connect OntologyGame;

select * from questions;

A list of questions should be displayed on the screen like figure A.1.



C:\WINDOWS\system32\cmd.exe - mysql -h localhost -u root

```
mysql> INSERT INTO Taboo (QuestionID,Taboo ) VALUES(15, 'dairy product');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Taboo (QuestionID,Taboo ) VALUES(16, 'cheese');
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.06 sec)

mysql> select * from questions;
+------------+---------------------+----------------------+------------+-------
-+-------------+------+
| QuestionID | MainQuestion        | QuestionDescription  | OntologyID | PassNo
 | RandomOrder | flag |
+------------+---------------------+----------------------+------------+-------
-+-------------+------+
|          1 | animal              | -------- is a kind of |          1 |      0
 |           7 | Y    |
|          2 | reptile             | -------- is a kind of |          1 |      0
 |          22 | Y    |
|          3 | mammal              | -------- is a kind of |          1 |      0
 |           2 | Y    |
|          4 | crocodile           | -------- is a kind of |          1 |      0
 |          84 | Y    |
|          5 | cat                 | -------- is a kind of |          1 |      0
 |          28 | Y    |
|          6 | domestic cat        | -------- is a kind of |          1 |      0
 |          42 | Y    |
|          7 | publication         | -------- is a kind of |          2 |      0
 |          11 | Y    |
|          8 | article             | -------- is a kind of |          2 |      0
 |          64 | Y    |
|          9 | journal             | -------- is a kind of |          2 |      0
 |          37 | Y    |
|         10 | musical work        | -------- is a kind of |          3 |      0
 |          55 | Y    |
|         11 | oeuvre              | -------- is a kind of |          3 |      0
 |          29 | Y    |
|         12 | orchestral work     | -------- is a kind of |          3 |      0
 |          81 | Y    |
|         13 | concerto            | -------- is a kind of |          3 |      0
 |          24 | Y    |
|         14 | food classification | -------- is a kind of |          4 |      0
 |          72 | Y    |
|         15 | proteins            | -------- is a kind of |          4 |      0
 |          49 | Y    |
|         16 | dairy product       | -------- is a kind of |          4 |      0
 |          18 | Y    |
|         17 | cheese              | -------- is a kind of |          4 |      0
 |           2 | Y    |
+------------+---------------------+----------------------+------------+-------
-+-------------+------+
17 rows in set (0.08 sec)

mysql>
```

Figure A.1: Database Installation Test

A.2.2 Running the Game

Game Server

After creating the database schema do the following:

1. Ensure that the database is still running by either ensuring that the command mysqld is still running or by typing the following command in the command line/shell.
   mysql -h host -u username -p password

   or if you do not have a password

mysql -h host -u username

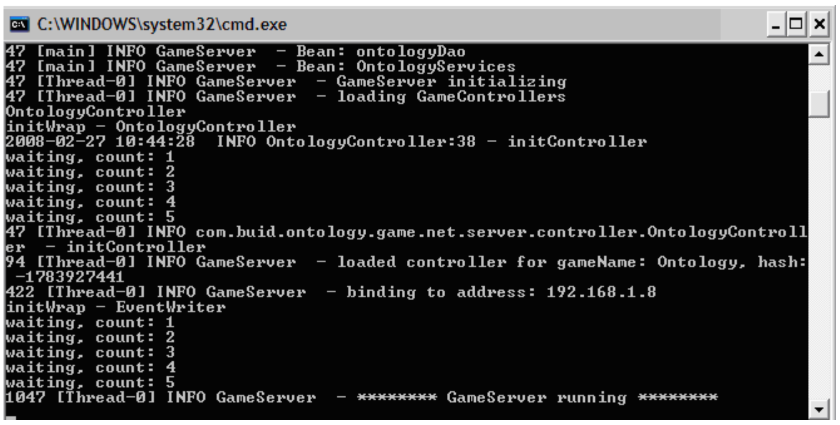
****host** is the host name where the database is installed e.g. localhost.
****username** is the username default root.
****password** is the password default empty.

If there were no errors connecting to the database, then it indicates that the database is running.

2. Start the Game Server

   a. If you are using Windows then run the startServer.batch file. The file can be found in Prototype/Server/startServer.batch directory
   b. If you are using Linux then run the startServer.sh file. The file can be found in Prototype/Server/startServer.sh directory

If the Game Server starts successfully, you will see the same output as figure A.2.



```
47 [main] INFO GameServer  - Bean: ontologyDao
47 [main] INFO GameServer  - Bean: OntologyServices
47 [Thread-0] INFO GameServer  - GameServer initializing
47 [Thread-0] INFO GameServer  - loading GameControllers
OntologyController
initWrap - OntologyController
2008-02-27 10:44:28  INFO OntologyController:38 - initController
waiting, count: 1
waiting, count: 2
waiting, count: 3
waiting, count: 4
waiting, count: 5
47 [Thread-0] INFO com.buid.ontology.game.net.server.controller.OntologyControll
er  - initController
94 [Thread-0] INFO GameServer  - loaded controller for gameName: Ontology, hash:
 -1783927441
422 [Thread-0] INFO GameServer  - binding to address: 192.168.1.8
initWrap - EventWriter
waiting, count: 1
waiting, count: 2
waiting, count: 3
waiting, count: 4
waiting, count: 5
1047 [Thread-0] INFO GameServer  - ******** GameServer running ********
```

Figure A.2: Game Server Running Test

Web Server

After running the game server do the following:

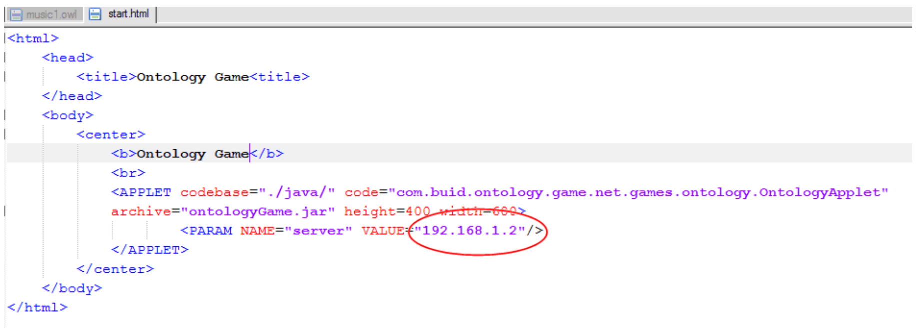
1. Edit the start.html. Change the IP address to the server IP (i.e. the IP of the machine where you are doing the installation) as shown in figure A.3
   ****start.html** can be found in Prototype/Client/start.html directory

2. Go to the following directory in tomcat
   C:/Program Files/Apache Software Foundation/Tomcat 5.5/webapps

3. Create a folder with the name **"game"**. Notice that the folder name is case–sensitive. So make sure that the name is all in small letters.

4. Place the start.html and java folder in the directory you just created i.e. **game** directory.
   ****start.html** and java directory can be found in Prototype/Client directory

5. Start apache tomcat



```
<html>
    <head>
        <title>Ontology Game<title>
    </head>
    <body>
        <center>
            <b>Ontology Game</b>
            <br>
            <APPLET codebase="./java/" code="com.buid.ontology.game.net.games.ontology.OntologyApplet"
            archive="ontologyGame.jar" height=400 width=600>
                <PARAM NAME="server" VALUE="192.168.1.2"/>
            </APPLET>
        </center>
    </body>
</html>
```

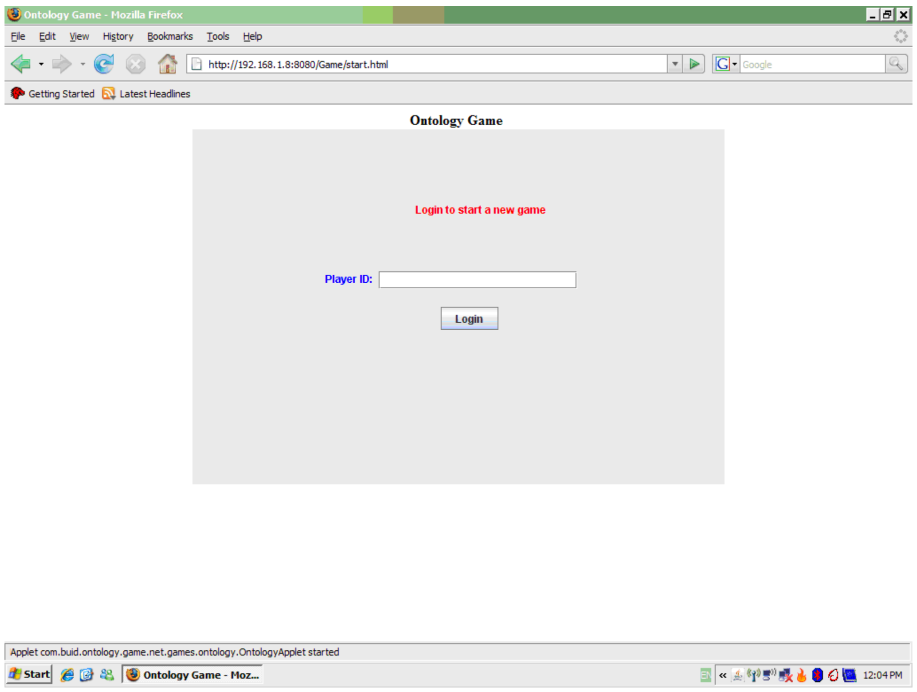
Figure A.3: Web Server IP Address

A.2.3 Testing the Game

In order to ensure that the game is running fine do the following:

1. Go to another machine **(Very important)**
2. Open a window using Mozilla Firefox browser
3. Access the following URL
   http://<server IP>:8080/game/start.html

   ****Server IP** is the IP of the machine where you installed the game.
4. You should be able to see the game like in figure A.4



Ontology Game

Login to start a new game

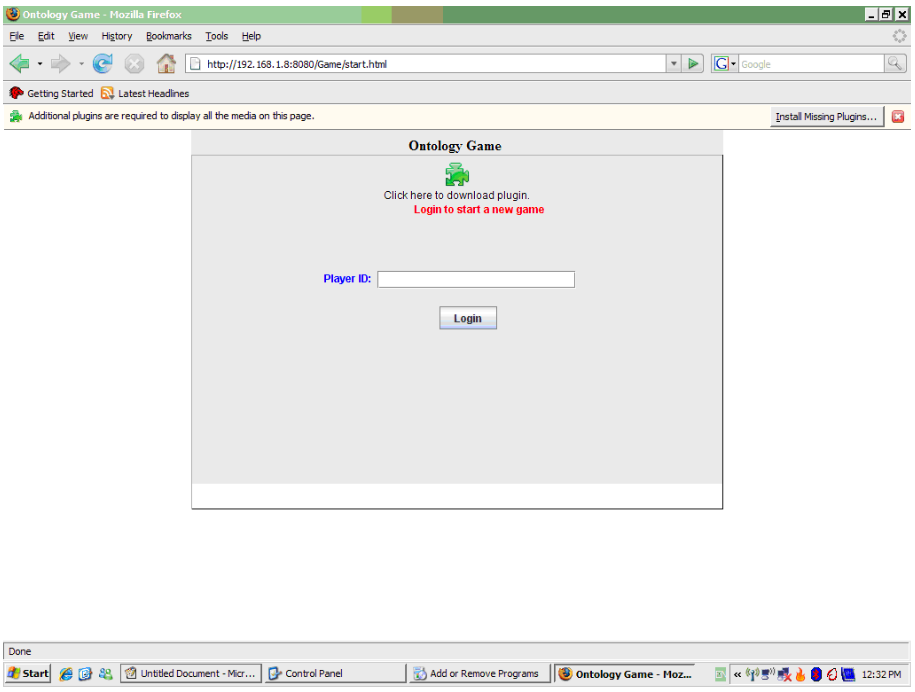
Player ID:

Login

Applet com.buid.ontology.game.net.games.ontology.OntologyApplet started

Figure A.4: Login Screen

5. If your window looks like figure A.5. Then if you are connected to the internet click on the **"Click here to download plugin"**, otherwise install sun jre 1.5 in this machine. Then try to access the URL again.


Ontology Game - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.1.8:8080/Game/start.html
Getting Started Latest Headlines
Install Missing Plugins...
Done
Start
Untitled Document - Micr...
Control Panel
Add or Remove Programs
Ontology Game - Moz...
12:32 PM

Additional plugins are required to display all the media on this page.

Ontology Game

Click here to download plugin.
Login to start a new game

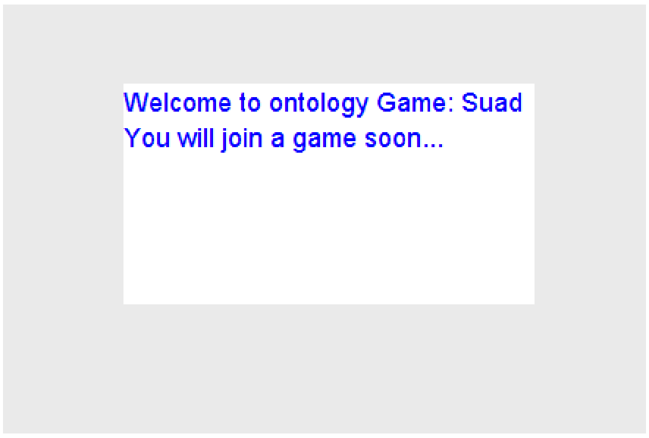
Player ID:

Login

Figure A.5: Plugin Required

6. If you are facing any other problem then, please ensure that you preformed all the steps, ensure that there is no exception thrown from the game server by looking at the server command line/shell or the log in the directory Prototype/Server/game.log and contact the author.

7. Login to the game providing any Player ID. You should see a screen like the one in figure A.6 and then a screen like in figure A.7.



Welcome to ontology Game: Suad
You will join a game soon...

Figure A.6: Successful Login Screen

8. Go to another machine and perform the same steps from 1 to 7. Ensure that you are providing a different Player ID than the one you used in the previous machine.

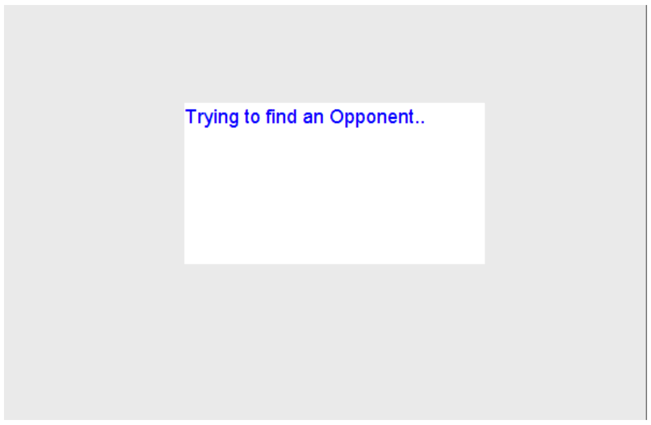
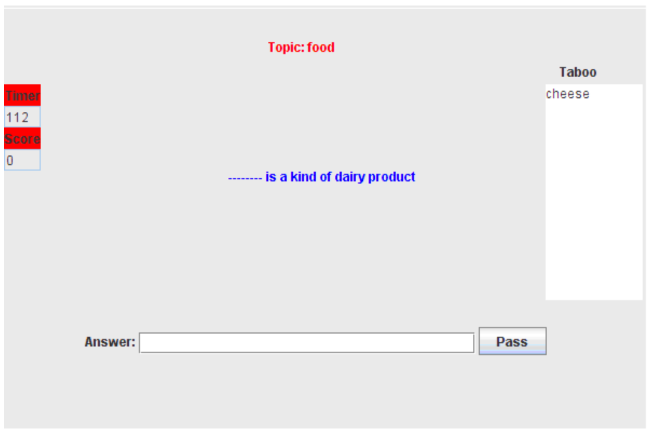

Trying to find an Opponent..

Figure A.7: Finding Opponent Screen

9. The screen in both of the machine should be identical. They should look like figure A.8


Topic: food

Taboo

cheese

Timer

112

Score

0

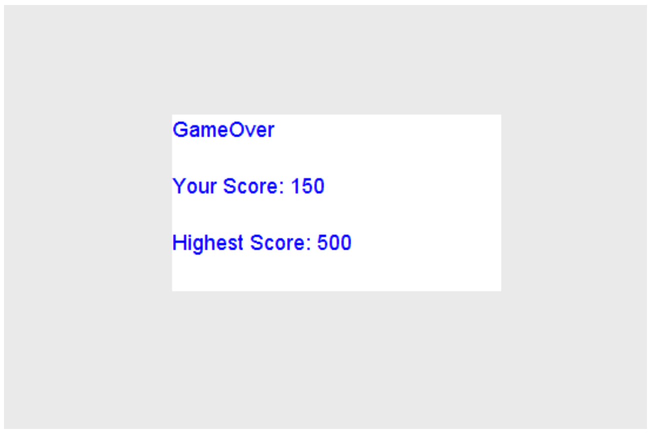
-------- is a kind of dairy product

Answer:

Pass

Figure A.8: Game Screen

10. Wait for two minutes for the game to get over and then your screen should look like figure A.9


GameOver

Your Score: 150

Highest Score: 500

Figure A.9: Gameover Screen

11. If any of the steps 9 or 10 fails and there was no exception on the game server, then restart the game server by stopping and starting the **startServer** script and try again.

A.2.4 Experiment Guidelines

Once you ensure that the game is running fine, you are ready to start the experiment. Following are the steps you need to follow to run the experiment.

1. Change the <Server IP> in the section "Playing the Game" to the IP of the machine where you installed the game.
2. Provide the "For the Volunteers" section which includes the **"Game Guidelines"** and **"Playing the Game"** sections to all the volunteers
3. Provide **Mozilla FireFox** and the **Sun jre 1.5 or later** setups to the volunteers as they need to have these software installed before playing the game.
4. Provide the materials in step 2 and 3 at least **one day** before running the experiment. So, that the volunteers can have time to read and install the prerequisites software
5. Ask all the volunteers to play the game at a predefined time.
6. The volunteers can play the game **N number** of times.
7. Once the experiment is over. Please ask the volunteers to provide their feedback about the game by filling up the survey which can be found at this location:

   http://www.buid.ac.ae/survey/ontology/

   A copy of the survey is provided in the "Ontology Game Survey" section. This copy can be used in case the URL can not be found or accessed. Please send it to all the volunteers and asked them to email it back to you and you can then email all the surveys to the author.
8. Export the database as mentioned in section A.2.6 and send it back to the author.

A.2.5 Troubleshooting the Game

If at any point in time the game was behaving in an unexpected way, then do the following:

1. Go to the machine where you installed that game server
2. Check the game server command line/shell i.e. the startServer window for any exception. You can also check the log file **"Prototype/Server/game.log"** for any exception
3. If there was any exception, then report it to the author.

4. If there is no exception, then close the command line/shell of the **startServer** and start it again.

   ** **startServer** can be found in the directory Prototype/Server/

Always ensure that the database is running by either ensuring that the command mysqld is still running or by typing the following command in the command line/shell.

```
mysql -h host -u username -p password
```

or if you do not have a password

```
mysql -h host -u username
```

** **host** is the host name where the database is installed e.g. localhost
** **username** is the username default root
** **password** is the password default empty.

If there were no errors connecting to the database, then it indicates that the database is running. If the database was not running, then open new command line/shell and type the command **mysqld**.

A.2.6 The Game output

Once the experiment is over, please export the database and send it back to the author. In order to do so do the following:

1. Run the database server if it is not running by opening new command line/shell and typing the command **mysqld**
2. Open another command line/shell
3. Type the following command

   ```
   mysqldump -u username -p password OntologyGame > FILE.sql
   ```

   or if you do not have a password

   ```
   mysqldump -u username OntologyGame > FILE.sql
   ```

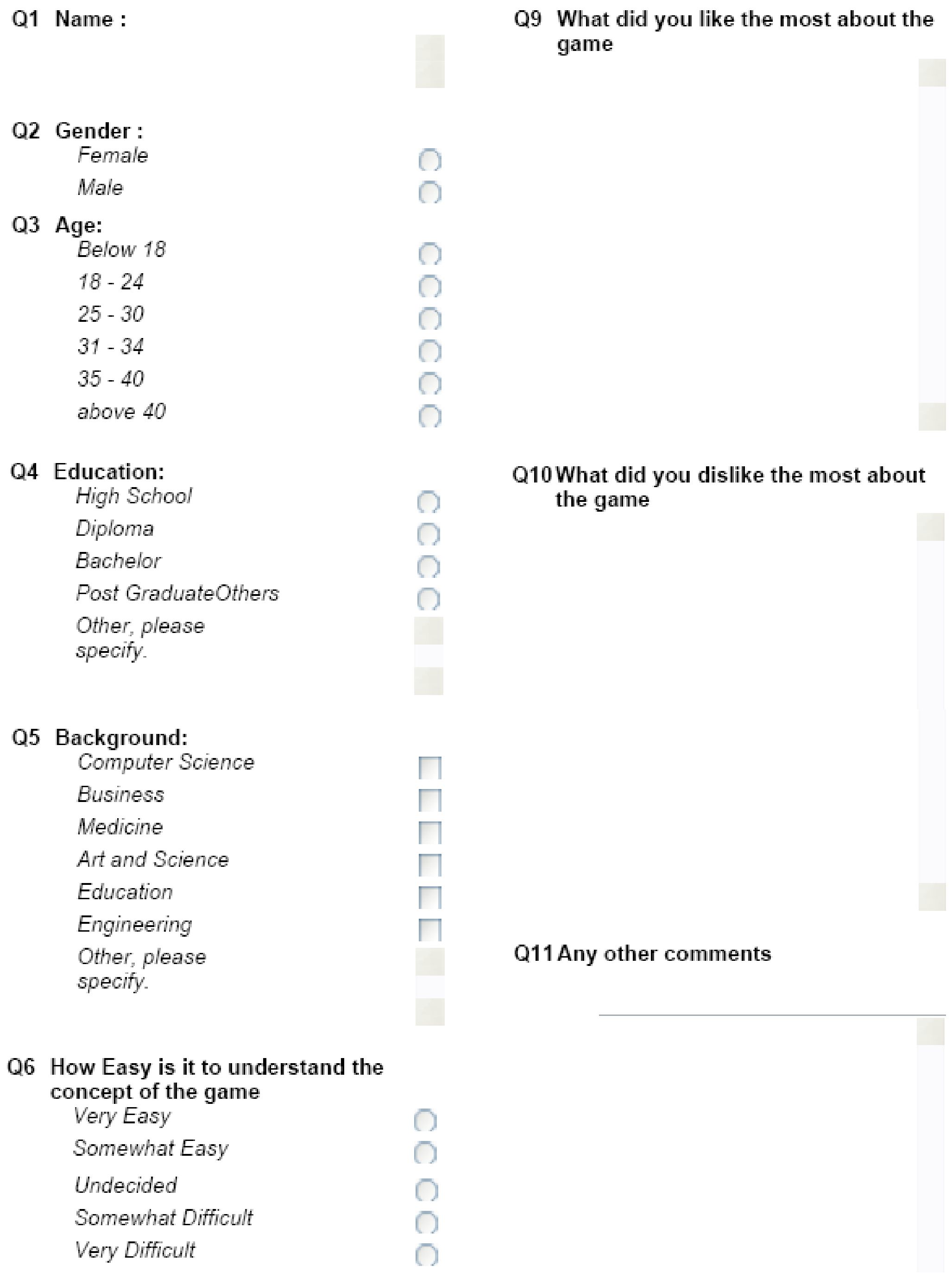
   ** **FILE.sql** new file that will be created in the current directory of the command line
   ** **username** is the username default root
   ** **password** is the password default empty.

4. Send the **FILE.sql** to the author

A.2.7 Ontology Game Survey

This survey is designed to evaluate the ontology game that is intended to speed up the process of building ontologies and to involve a large community of people in building the ontologies. Please answer all the questions.

Thank you for taking the time to participate in this survey. (This survey is strictly blind to individual participants.)



Q1 Name :

Q2 Gender :
- *Female*
- *Male*

Q3 Age:
- *Below 18*
- *18 - 24*
- *25 - 30*
- *31 - 34*
- *35 - 40*
- *above 40*

Q4 Education:
- *High School*
- *Diploma*
- *Bachelor*
- *Post GraduateOthers*
- *Other, please specify.*

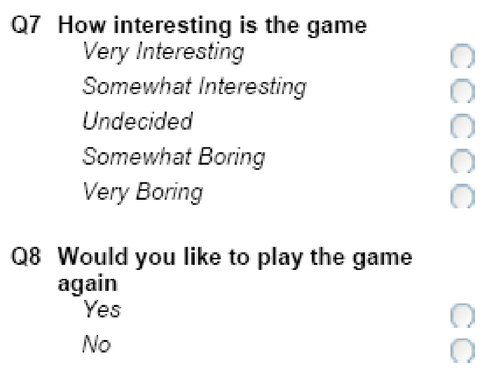
Q5 Background:
- *Computer Science*
- *Business*
- *Medicine*
- *Art and Science*
- *Education*
- *Engineering*
- *Other, please specify.*

Q6 How Easy is it to understand the concept of the game
- *Very Easy*
- *Somewhat Easy*
- *Undecided*
- *Somewhat Difficult*
- *Very Difficult*

Q9 What did you like the most about the game

Q10 What did you dislike the most about the game

Q11 Any other comments



Q7 How interesting is the game
- Very Interesting ○
- Somewhat Interesting ○
- Undecided ○
- Somewhat Boring ○
- Very Boring ○

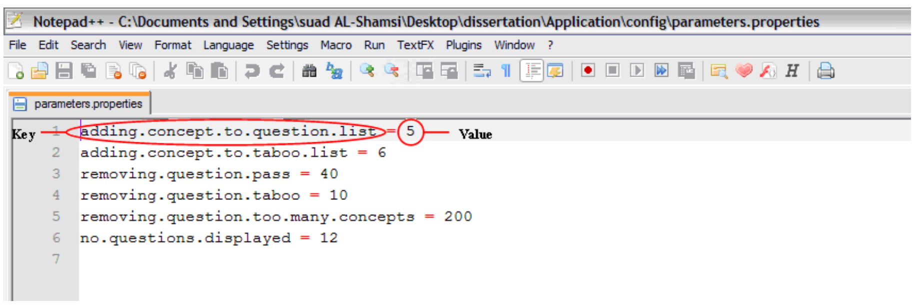
Q8 Would you like to play the game again
- Yes ○
- No ○

A.2.8 Configure the Game's Parameters

This section explains how to configure the game's parameters. These parameters should not be changed without the author's instructions as it might have a negative effect on the experiment's output.

Following are the steps to configure the game's parameters:

1. Navigate to the following directory **Prototype/Server/config**
2. Open the file **parameters.properties**
3. The file contains a key/value pairs as in figure A.10


Notepad++ - C:\Documents and Settings\suad AL-Shamsi\Desktop\dissertation\Application\config\parameters.properties
File Edit Search View Format Language Settings Macro Run TextFX Plugins Window ?
parameters.properties
Key
Value

```
adding.concept.to.question.list = 5
adding.concept.to.taboo.list = 6
removing.question.pass = 40
removing.question.taboo = 10
removing.question.too.many.concepts = 200
no.questions.displayed = 12
```

Figure A.10: Configure Game Parameters

4. Modify the parameters value as instructed by the author
5. Restart the game server for the modifications to take place. To restart the game server close the command line/shell of the **startServer** and start it again.
   ** **startServer** can be found in the directory Prototype/Server/

A.3 For the Volunteers

A.3.1 Game Guidelines

Introduction:

An ontology is a shared understanding of some domains of interest. The purpose of the game is to speed up the process of building ontologies.

The Ontology Game is a multiplayer online game. Once you login to the game, you will be matched with an anonymous partner i.e. another online player. You and your partner will be presented with questions, and will need to write down all answers you can think of. You cannot see what your partner is typing, but if both of you write the same answer, you will earn points. You will have the option to pass if the question presented is hard. The game will last for two minutes only.

Game Snapshots:

Figure A.11 represents the main screen of the game.

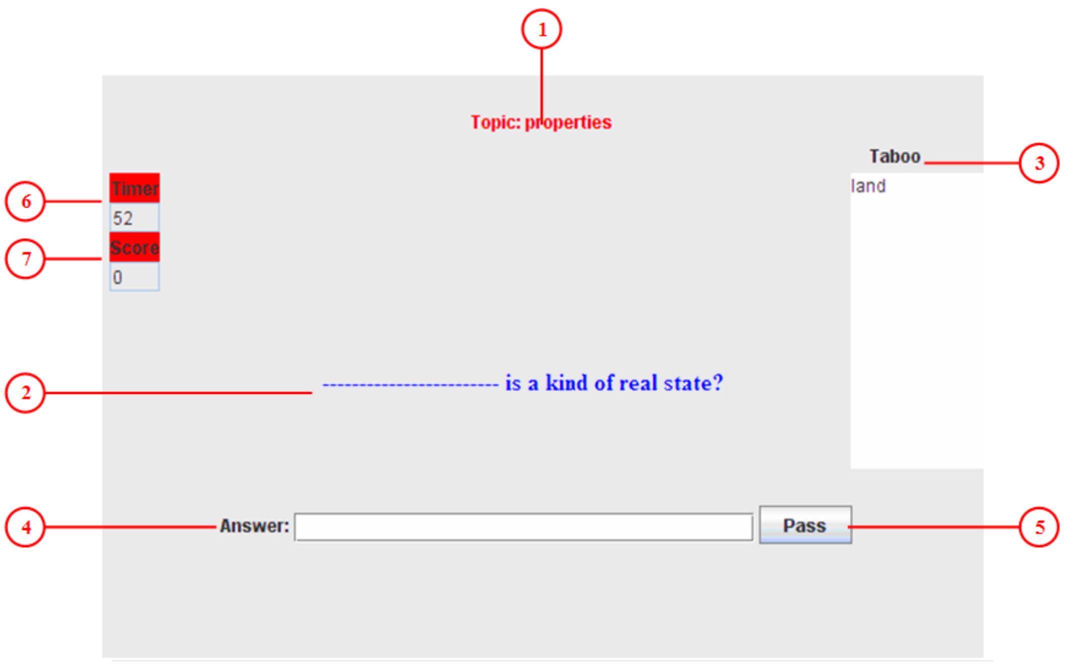


Figure A.11: The Main Game Screen

Game Components:

Here are few things that you need to know about the game before playing it:

1. **Topic:** This represents the context of the question being asked. Before reading the question have a look at the topic to increase the chances of earning points.

2. **Question:** This is the question you need to answer.

3. **Taboo:** This represents the list of words that you are not allowed to provide as an answer.

4. **Answer:** Type your answer inside the **box** and then **hit enter**. Keep typing answers as much as you can think of to increase your chances of earning more points. Provide only one answer at a time and then hit enter. Then provide another answer and hit enter. Try to provide general answers. For instance for the question "**——– is a kind of vehicle?**", the answers can be **"two wheel drive"** and **"four wheel drive"**.

5. **Pass:** If the question is hard and you want to change the question then click on the **Pass Button**. Your partner will receive a message like in figure A.12. If your partner agrees to pass by hitting the **Pass Button** as well, the question will change.

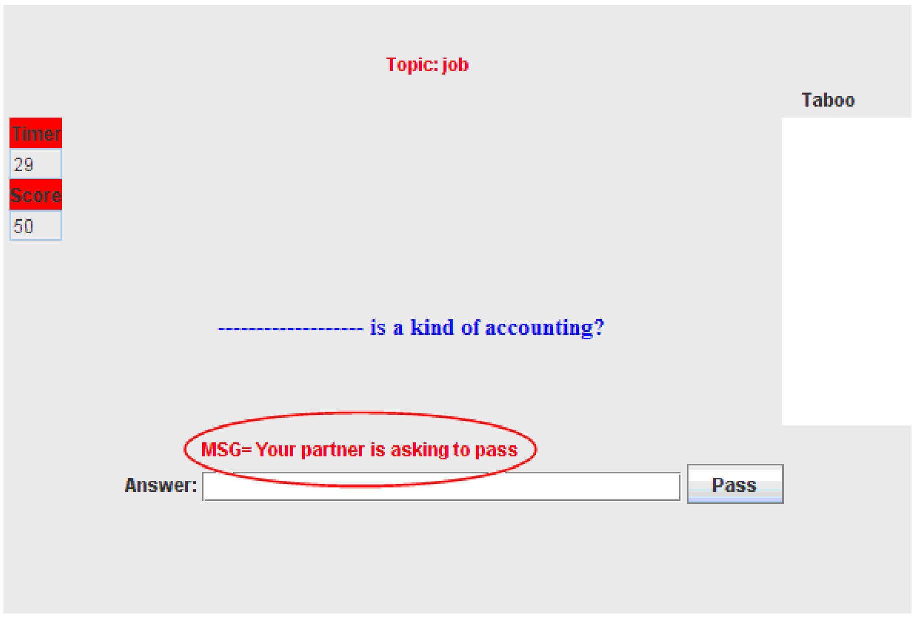

Topic: job
Taboo
Timer
29
Score
50
------------------ is a kind of accounting?
MSG= Your partner is asking to pass
Answer:
Pass

Figure A.12: Pass Request Screen

6. **Time:** Displays the remaining time of the game in seconds.

7. **Score:** Displays your score so far.

A.3.2 Playing the Game

Prerequisites:

In order to play the game, install the following software:

1. Sun jre 1.5 or later

2. Mozilla FireFox

Instructions:

1. Access the game URL **http://192.168.10.148:9080/game/start.html**
2. Your screen should look like figure A.13. If not please contact your administrator/staff who sent the game

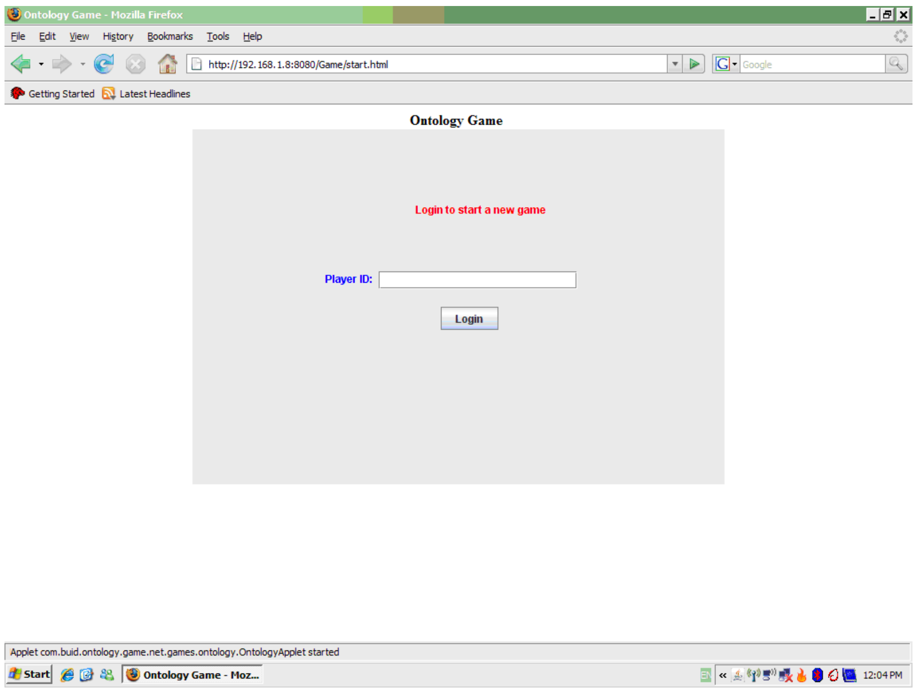
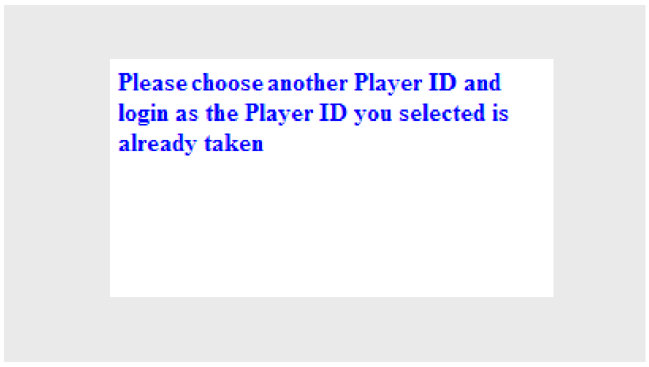

Ontology Game - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.1.8:8080/Game/start.html
Getting Started Latest Headlines
Ontology Game
Login to start a new game
Player ID:
Login
Applet com.buid.ontology.game.net.games.ontology.OntologyApplet started
Start
Ontology Game - Moz...
12:04 PM

Figure A.13: Login Screen

3. Enter your user name in the Player ID. Try to use a unique name. If somebody else has taken the name before you, you will get a message like in figure A.14. If it was the case then just fresh the page and use another name



Please choose another Player ID and login as the Player ID you selected is already taken

Figure A.14: Failure Login Screen

4. Once you logged in successfully, your screen will look like figure A.15.
5. If there is a free player, the game will start and your screen will look like figure A.16.

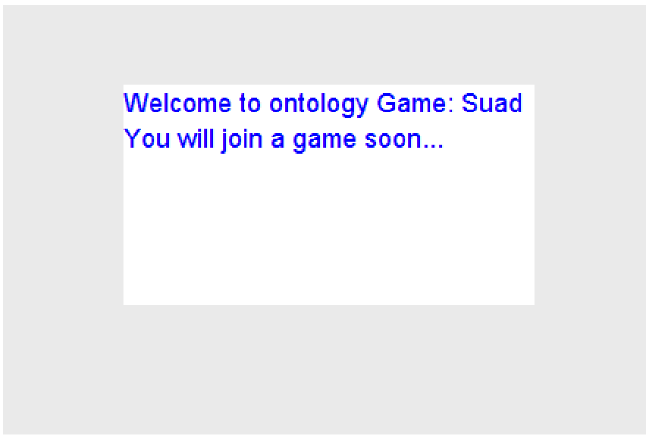

Welcome to ontology Game: Suad
You will join a game soon...

Figure A.15: Successful Login Screen

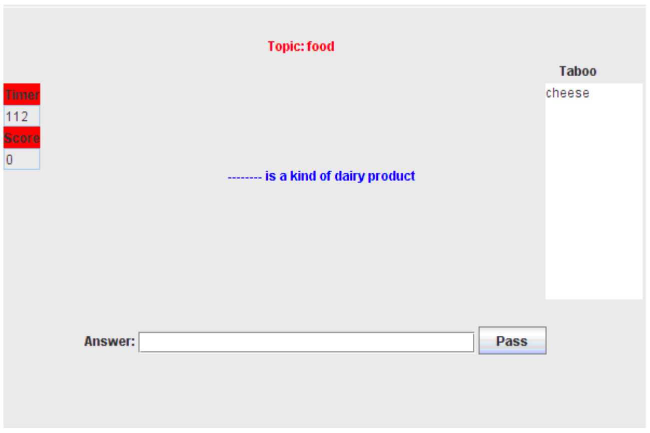


Figure A.16: Game Screen

Remember the Game Rules:

a. You will have two minutes to play.

b. Try to answer as many questions as you can.

c. When you will provide an answer hit **enter**

d. Avoid the answers that are listed in the taboo.

e. Keep your eyes on the question while answering

f. Read the topic before you start answering

g. Click the **Pass Button** to pass the question

6. If there are no free players then a message will be prompted to you stating that it is trying to find a partner like in figure A.17. If it is the case, then just wait for some time. **Please do not close the browser**.

A.3.3 Very Important Tips:

1. Please avoid closing the browser while playing the game.

2. Do not login more than once from the same machine even if you are using different user name.

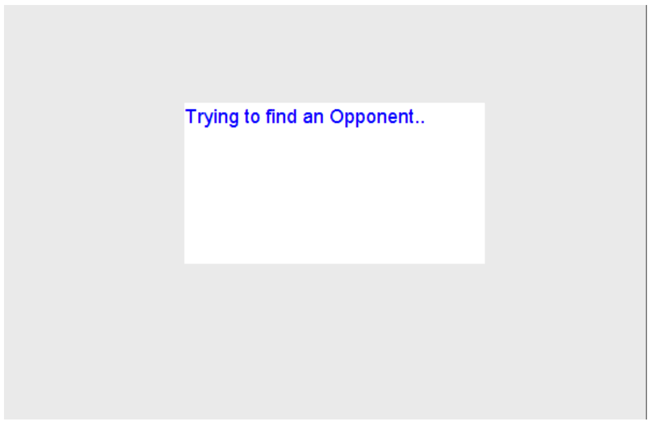

Trying to find an Opponent..

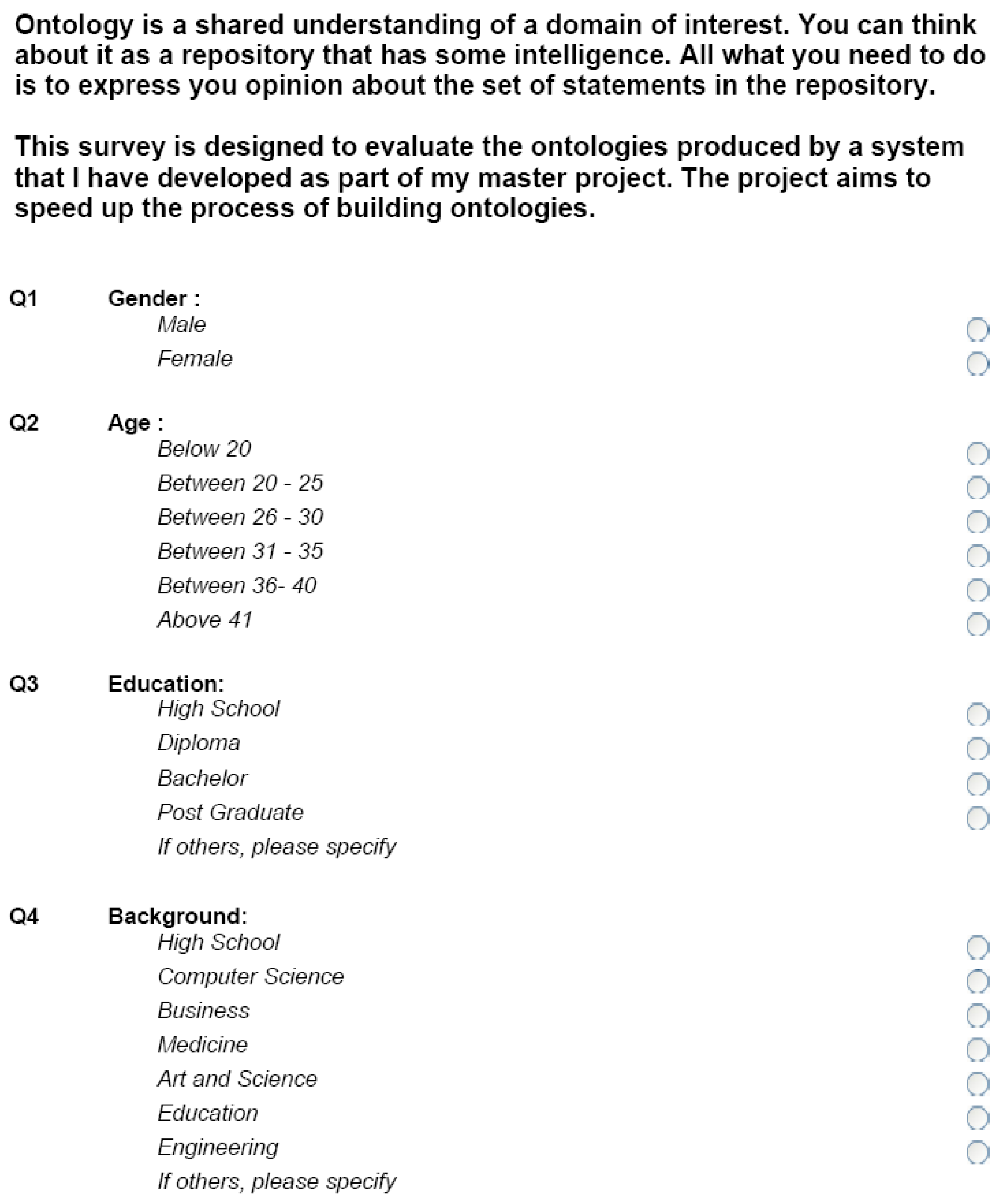
Figure A.17: Finding Opponent Screen

Please provide me with your valuable feedback by filling the survey looked at http://www.buid.ac.ae/survey/ontology/

I really appreciate your help and support.

Appendix B

Ontology Evaluation Survey



Ontology is a shared understanding of a domain of interest. You can think about it as a repository that has some intelligence. All what you need to do is to express you opinion about the set of statements in the repository.

This survey is designed to evaluate the ontologies produced by a system that I have developed as part of my master project. The project aims to speed up the process of building ontologies.

Q1 **Gender :**

- *Male* ○
- *Female* ○

Q2 **Age :**

- *Below 20* ○
- *Between 20 - 25* ○
- *Between 26 - 30* ○
- *Between 31 - 35* ○
- *Between 36- 40* ○
- *Above 41* ○

Q3 **Education:**

- *High School* ○
- *Diploma* ○
- *Bachelor* ○
- *Post Graduate* ○
- *If others, please specify*

Q4 **Background:**

- *High School* ○
- *Computer Science* ○
- *Business* ○
- *Medicine* ○
- *Art and Science* ○
- *Education* ○
- *Engineering* ○
- *If others, please specify*

Q6 **For each of the below statement please choose your Degree of Agreement**

	1 - Totally agree	*2*	*3*	*4*	*5 - Totally disagree*
Plane is a kind of **Aircraft**	○	○	○	○	○
Butter is a kind of **Diary Product**	○	○	○	○	○
Lizard is a kind of **Reptile**	○	○	○	○	○
BMW is a kind of **Car**	○	○	○	○	○
Dog is a kind of **Mammal**	○	○	○	○	○
Four Wheel is a kind of **Vehicle**	○	○	○	○	○
Fish is a kind of **Meat**	○	○	○	○	○
Bus is a kind of **Vehicle**	○	○	○	○	○
Monkey is a kind of **Mammal**	○	○	○	○	○
X5 is a kind of **BMW**	○	○	○	○	○
Snake is a kind of **Reptile**	○	○	○	○	○
Meat is a kind of **Proteins**	○	○	○	○	○
Truck is a kind of **Vehicle**	○	○	○	○	○
Camel is a kind of **Mammal**	○	○	○	○	○
Toyota is a kind of **Car**	○	○	○	○	○
Milk is a kind of **Diary Product**	○	○	○	○	○
Cow is a kind of **Mammal**	○	○	○	○	○
Car is a kind of **Wheeled Vehicle**	○	○	○	○	○

Thank you for taking the time to participate in this survey.

Appendix C

Experiment Result

C.1 Knowledge Collected

Table C.1 represents the various answers provided by the volunteers and the total number of agreements on these answers for the "Food Classification" ontology, while table C.2 represents the various answers provided by the volunteers and the total number of agreements on these answers for the "Animal" ontology.

Sub–class	**Super–class**	**Agreements count**
Egg	Proteins	1
Milk	Dairy product	6
Craft	Cheese	1
Milk	Proteins	3
Meat	Proteins	3
Yoghurt	Dairy product	1
Butter	Dairy product	3
Fish	Meat	1

Table C.1: Experiment Knowledge Collected for the Food Classification Ontology

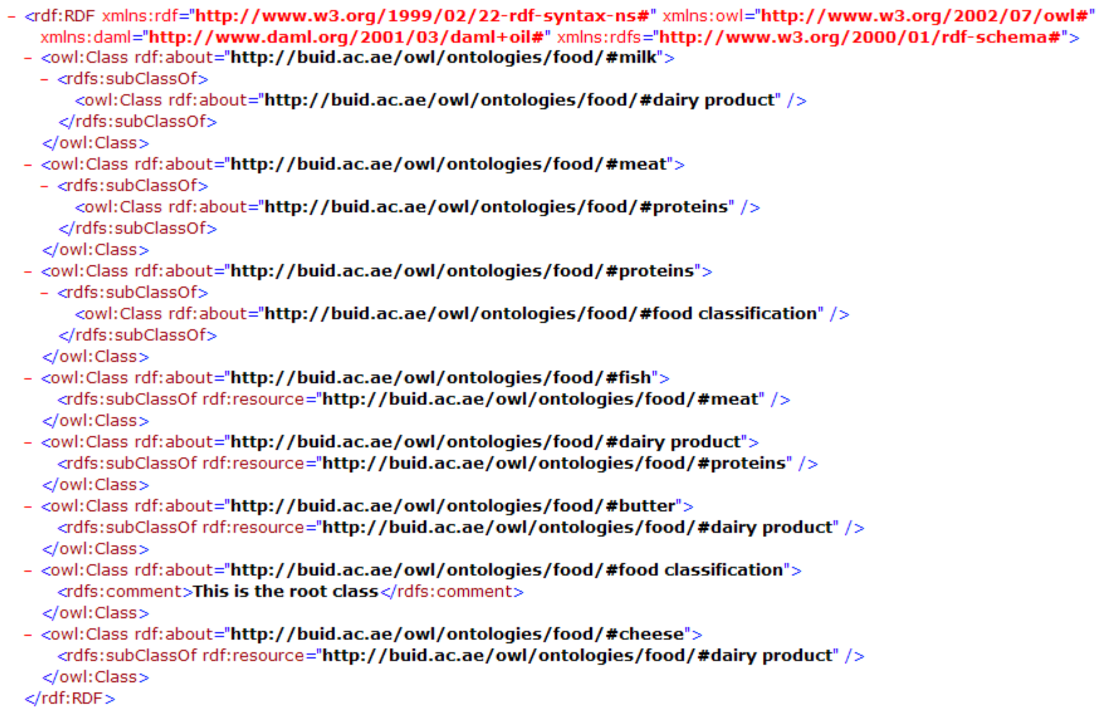
C.2 Ontologies Produced

Figure C.1 represents the OWL file generated for the "Food Classification" ontology. The hierarchical relationships between the concepts in the ontology are shown in figure C.2. The concepts that are added by the volunteers are represented in red boxes.

Sub–class	**Super–class**	**Agreements count**
Dog	Mammal	5
Turtle	Reptile	1
Lizard	Reptile	2
Wild cat	Cat	1
Lizard	Animal	1
Tiger	Mammal	1
Cow	Mammal	2
Camel	Mammal	2
Dog	Animal	5
Cat	Animal	9
Elephant	Mammal	1
Monkey	Mammal	2
Whale	Mammal	1
Snake	Reptile	2
Lion	Animal	1

Table C.2: Experiment Knowledge Collected for the Animal Ontology

Figure C.3 represents the OWL file generated for the “Animal Classification” ontology. The hierarchical relationships between the concepts in the ontology are shown in figure C.4. The concepts that are added by the volunteers are represented in red boxes.



```
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/food/#milk">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/food/#dairy product" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/food/#meat">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/food/#proteins" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/food/#proteins">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/food/#food classification" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/food/#fish">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/food/#meat" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/food/#dairy product">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/food/#proteins" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/food/#butter">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/food/#dairy product" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/food/#food classification">
      <rdfs:comment>This is the root class</rdfs:comment>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/food/#cheese">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/food/#dairy product" />
    </owl:Class>
  </rdf:RDF>
```

Figure C.1: Food Classification Ontology in OWL Format

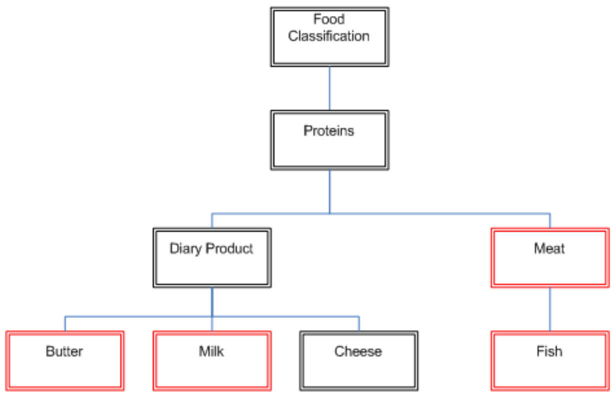
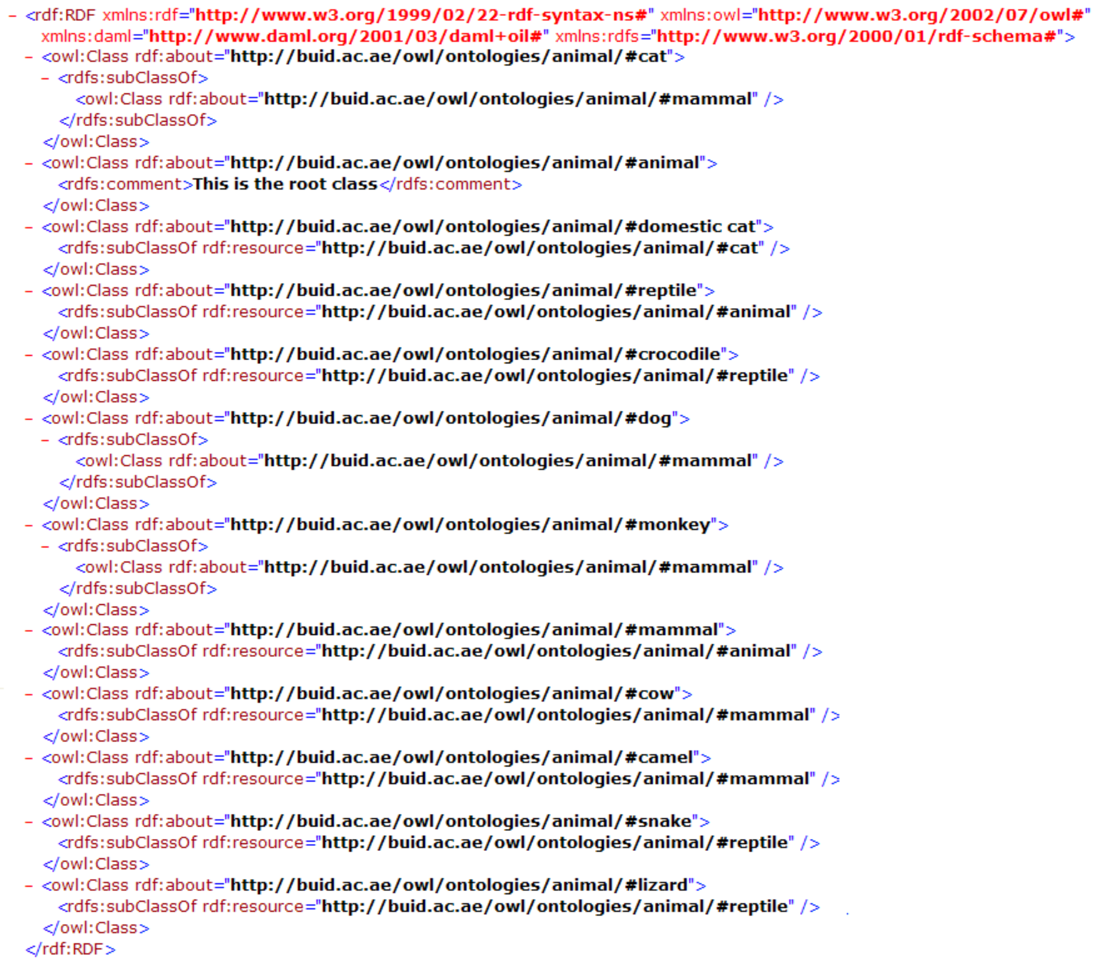


Figure C.2: Food Classification Ontology hierarchy



```
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#cat">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#mammal" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#animal">
      <rdfs:comment>This is the root class</rdfs:comment>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#domestic cat">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/animal/#cat" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#reptile">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/animal/#animal" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#crocodile">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/animal/#reptile" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#dog">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#mammal" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#monkey">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#mammal" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#mammal">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/animal/#animal" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#cow">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/animal/#mammal" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#camel">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/animal/#mammal" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#snake">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/animal/#reptile" />
    </owl:Class>
  - <owl:Class rdf:about="http://buid.ac.ae/owl/ontologies/animal/#lizard">
      <rdfs:subClassOf rdf:resource="http://buid.ac.ae/owl/ontologies/animal/#reptile" />
    </owl:Class>
  </rdf:RDF>
```

Figure C.3: Animal Ontology in OWL Format

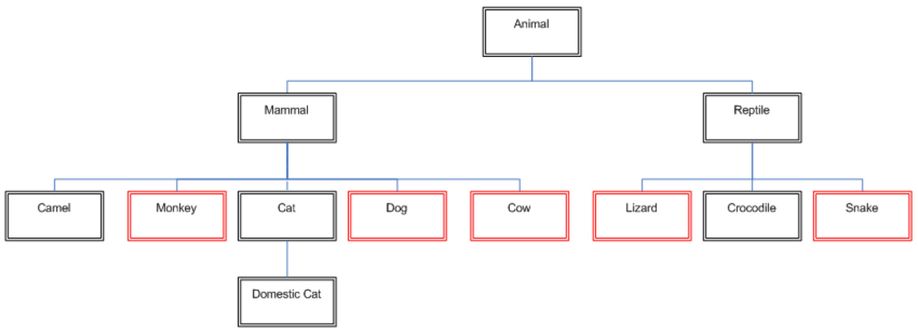


Figure C.4: Animal Ontology hierarchy

Bibliography

[1] Human–based computation. Online.

[2] Mindpixel Project. Online, June 2007.

[3] Alexander Maedche and Steffen Staab. Measuring Similarity between Ontologies. In *Proceedings of the European Conference on Knowledge Acquisition and Management (EKAW)*, volume 2473, pages 251–263, Madrid, Spain, October 2002. Springer LNCS.

[4] Alexandre Passant. Using Ontologies to Strengthen Folksonomies and Enrich Information Retrieval in Weblogs: Theoretical background and corporate use-case. March 2007.

[5] Céline Van Damme and Martin Hepp and Katharina Siorpaes. FolksOntology: An Integrated Approach for Turning Folksonomies into Ontologies. In *Bridging the Gep between Semantic Web and Web 2.0 (SemNet 2007)*, Proceedings of the ESWC 2007 Workshop, pages 71–84, Innsbruck, Austria, June 2007.

[6] David Brackeen, Bret Barker, Laurence Vanhelsuwé. *Developing Games in Java*. New Riders Publishing, August 2003.

[7] Douglas B. Lenat. CYC: a large-scale investment in knowledge infrastructure. *Commun. ACM*, 38(11):33–38, 1995.

[8] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. The MIT Press, London, England, 2004.

[9] Henry Lieberman and Dustin A Smith and Alea Teeters. Common Consensus: A Web-based Game for Collecting Commonsense Goals. In *Commonsense Workshop at the ACM International Conference on Intelligent User Interfaces (IUI-07)*, Honolulu, January 2007.

[10] Jan Henke. Towards a Usable Group Editor for Ontologies. In *International Semantic Web Conference*, pages 978–979, 2006.

[11] Janez Brank and Marko Grobelnik and Dunja Mladenić. A survey of ontology evaluation techniques. In *SIKDD 2005 at multiconference IS 2005*, Ljubljana, Slovenia, October 2005.

[12] Katharina Siorpaes and Martin Hepp. myOntology: The Marriage of Ontology Engineering and Collective Intelligence. In *Bridging the Gep between Semantic Web and Web 2.0 (SemNet 2007)*, Proceedings of the ESWC 2007 Workshop, pages 127–138, Innsbruck, Austria, June 2007.

[13] Katharina Siorpaes and Martin Hepp. OntoGame: Towards Overcoming the Incentive Bottleneck in Ontology Building. In *Proceedings of the 3rd International IFIP Workshop On Semantic Web & Web Semantics (SWWS '07) co-located with OTM Federated Conferences*, volume 4806 of *OTM 2007 Workshops (2)*, pages 1222–1232, Vilamoura, Portugal, November 2007. Springer LNCS.

[14] L. Lovász , J. Pelikán , K. Vesztergombi. *Discrete Mathematics: Elementary and Beyond*. Springer, New York, August 2003.

[15] Lucia Specia and Enrico Motta. Integrating Folksonomies with the Semantic Web. In *Proceedings of the 4th European Semantic Web Conference (ESWC2007)*, Innsbruck, Austria, 2007. Springer LNCS.

[16] Luis von Ahn. Games with a Purpose. *IEEE Computer*, 39(6):92–94, 2006.

[17] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 319–326, New York, NY, USA, 2004. ACM.

[18] Luis von Ahn and Mihir Kedia and Manuel Blum. Verbosity: a game for collecting common-sense facts. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 75–78, New York, NY, USA, 2006. ACM.

[19] Luis von Ahn and Ruoran Liu and Manuel Blum. Peekaboom: a game for locating objects in images. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 55–64, New York, NY, USA, 2006. ACM.

[20] Luis von Ahn and Shiry Ginosar and Mihir Kedia and Ruoran Liu and Manuel Blum. Improving accessibility of the web with a computer game. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 79–82, New York, NY, USA, 2006. ACM.

[21] Martin Hepp and Daniel Bachlechner and Katharina Siorpaes. OntoWiki: community-driven ontology engineering and ontology usage based on Wikis. In Dirk Riehle and James Noble, editor, *Int. Sym. Wikis*, Proceedings of the 2006 international symposium on Wikis, pages 143–144, New York, USA, 2006. ACM Press.

[22] Mike Uschold and Martin King. Towards a Methodology for Building Ontologies. In *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing*, Montreal, Canada, 1995.

[23] Natalya F. Noy and Deborah L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Technical report, Stanford University School of Medicine, 2001. This guide describes a common methodology for ontology–development based on declerative frame–based systems. Upshot: there is no single correct ontology for any domain.

[24] Natalya Fridman Noy and Mark A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 450–455, Austin, Texas, 2000. AAAI Press / The MIT Press.

[25] PeterBrown, DeniseBedford, JimDisbrow, JackTeller and SusanTurnbull. Ontology Summit 2007: Ontology, Taxonomy, Folksomony Distinctions. 2007.

[26] Push Singh. The Open Mind Common Sense Project. Online, January 2002.

[27] Simone Braun and Andreas Schmidt and Andreas Walter and Gabor Nagypal and Valentin Zacharias. Ontology Maturing: a Collaborative Web 2.0 Approach to Ontology Engineering. In *Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at the 16th International World Wide Web Conference (WWW2007)*, Banff, Alberta, Canada , 2007.

[28] Thomas Vander Wal. Folksonomy. Online, February 2007.

[29] Ying Ding and Dieter Fensel and Michel C. A. Klein and Borys Omelayenko. The semantic web: yet another hip? *Data Knowledge Engineering*, 41(2–3):205–227, 2002.

[30] York Sure and Jurgen Angele and Steffen Staab. OntoEdit: Guiding Ontology Development by Methodology and Inferencing. In *Proceedings of the first International Semantic Web Conference 2002 (ISWC 2002)*, pages 221–235, Sardinia, Italy, June 2002. Springer, LNCS 2342.