THE UNIVERSITY *of* LIVERPOOL

Ontology-based Fault Diagnosis for Power Transformers

Thesis submitted in accordance with the
requirements of the University of Liverpool
for the degree of Master of Philosophy

in

Electrical Engineering and Electronics

by

Dian Wang, B.Eng.

July 2011

Ontology-based Fault Diagnosis for Power Transformers

by

Dian Wang

To my parents

Acknowledgements

Firstly, a special acknowledgement should be extended to my supervisor Dr. W. H. Tang for his invaluable help, his patient discussion and support on my research project. His scientific belief and his persistent efforts constitute the essential part of the project, and his knowledge and passions in science exceptionally inspire and enrich my growth. This thesis would not have been completed without his help.

Many thanks to Prof. Q. H. Wu, for his professional guidance, his patience in supervision. His drive, enthusiasm, work and knowledge have triggered and nourished my intellectual maturity.

I also want to thank Dr. T. Y. Ji, Dr. D. Y. Shi, Mr. L. Wang and other colleagues from Intelligence Engineering and Automation Research Group, the University of Liverpool for their unreserved assistance with many different problems I have encountered throughout my studies. Particularly, many thanks to Dr. C. Ma who has given me many valuable suggestions on this thesis.

The University of Liverpool Department of Electrical Engineering and Electronics has provided excellent research facilities, making it possible for me to conduct my research.

I would extend my thanks to the University of Liverpool for the studentship (2008 – 2009).

I am greatly indebted to my parents, for their encouragement, support and understanding. With love, they have always been there for me, whenever good time or bad.

Finally, I am greatly indebted to my wife Mengya Hu, for her patience and understandings and her dedications to our small family in UK.

Abstract

Electrical engineers are frequently required to undertake tasks on condition assessment, fault diagnosis, operation decision-making and maintenance of power transformers, based on their knowledge, experience and expertise by comparing present and past measurement data. In some cases, it requires a team of experts in different areas and a huge amount of information has been produced and stored for such tasks. The volume of data has exceeded the capability of data analysis of engineers with limited individual knowledge, as these data are in association with complex and comprehensive concepts and knowledge of power system operations. Therefore, new techniques for knowledge representation, automated data analysis and decision-making are required, in order to reduce the need for human intervention in handling the complex data and individual knowledge.

Expert-system is widely used for transformer fault diagnosis, which is provided with strong pertinence yet the expansibility is comparatively weak. Pursuant to the diversified inference mechanism and knowledge library structure, the former problems on knowledge exchange in such systems could never be solved. The current diagnosis systems for transformer faults are mostly based on detecting variations of a transformer. Given no effective integration for the current methods of diagnosis, it is necessary to introduce a new system which can integrate a variety of diagnostic methods to enhance the diagnosis efficiency.

Ontology is a mechanism that describes concepts and their system relationships. An ontology-based knowledge representation has several attractive features and holds the fact that it focuses on the classification and constraints

of allowable taxonomies and definitions. The formal nature of ontology also enables the integration of data from heterogeneous sources.

In this thesis, ontology is employed to enhance the exchanging-study ability between heterogeneous systems as well as realizing authentic knowledge exchange. As the knowledge foundation of the whole system, an ontology knowledge library guarantees the realization of a higher-level knowledge exchange. In order to develop an ontology system for the fault diagnosis of power transformers, it is necessary to analyse numerous concepts and relationships exhibited for power transformers. This thesis proposes a power transformer fault diagnosis system with ontology, which is concerned as a part of power system ontology. This ontology provides a semantic model for knowledge representation and information management. It can be used to integrate a number of transformers diagnostic methods, such as transformer thermal condition monitoring and diagnosis, dissolved gas analysis, partial discharge analysis and frequency response analysis *etc.*

A new approach to transformer fault diagnosis is introduced in this thesis based on the idea of exchanging information with explicit, formal and machine accessible descriptions of meaning by using the Semantic Web. An ontology model is developed for accurate and efficient fault diagnosis for power transformers. Through the use of this model, various transformer faults diagnostic methods can be integrated to describe an inference among fault phenomena, fault sources and causes of faults. The proposed ontology model provides a dedicated semantic model for knowledge representation and information management concerning fault diagnosis of power transformers. Finally, a systematic summary is given. Challenges are discussed and future research work is suggested.

Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

AI	Artificial Intelligence
ANNs	Artificial neural networks
ANSI	American National Standards Institute
API	Application Programming Interface
BOT	Bottom-oil Temperature
DGA	Dissolved Gas Analysis
DL	Description Logics
ES	Expert System
FRA	Frequency Response Analysis
FT	Fourier Transform
HF	High Frequency
JDT	Java Development Tools
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
LF	Low Frequency
MF	Medium Frequency
NEMA	National Electrical Manufacture Association
OKBC	Open Knowledge Base Connectivity
OWL	Ontology Web Language
PD	Partial Discharge
PDA	Partial Discharge Analysis
RDF	Resource Description Framework

RDFs	Resource Description Framework Schema
TF	Thermal Fault
TM	Thermal Modelling
TOT	Top-oil Temperature
URI	Universal Resource Identifier
WT	Wavelet Transform
WTI	Winding Temperature Indicator
W3C	World Wide Web Consortium

Chapter 1

Introduction

This thesis is concerned with the development of fault diagnosis for power transformers using the ontology technologies. An overview of fault diagnosis for power transformers and a brief introduction to the ontology technologies are given at the beginning. The existing problems and deficiencies of using the conventional methodologies to handle the issues of fault diagnosis for power transformers are discussed in details in the following sections. Finally, a thesis outline is provided to give a clear view of the entire contents.

1.1 Background of Fault Diagnosis for Power Transformers

There are three functional segments in electric utility industry: generation, transmission, and distribution. Some devices, such as power transformers, on-load taps changers (OLTCs), circuit breakers, current transformers, station batteries, and other switch gear, play critical roles in the power transmission and distribution for ensuring the process of transporting electricity and delivering electricity from the generating devices to the ultimate consumers. Any faults of these devices could result in power outages, personal or environmental hazards, and economic loss. Therefore, critical assets of utilities, such as the transformers, should be monitored closely and continuously to make sure the

devices are properly working within its maximum uptime. The transformer is one of the most expensive elements in the power system with two main functions in the transmission and distribution network. On the one hand, the transformer can alter the voltage or current in an AC circuit, isolate circuits from each other and shift the apparent value of a capacitor, an inductor or a resistor. On the other hand, it enables to transmit electrical energy over great distance and to distribute it safely to the ultimate consumers. In many power stations, a power transformer is worth thousands to millions pounds. Any tiny faults may result in catastrophic damage of the whole system. So the correct functioning of the transformer and the diagnosis of power transformer fault is of great necessity and importance to system operations.

Transformer failure may owe to various causes during operation, and the most frequent causes are electrical disturbances, deterioration of insulation, lightning, inadequate maintenance, loose connections, moisture and overloading. The longer the transformer has been in use, the more possibility there would be to cause faults in the transformer. Because the faults are inevitable it is necessary to closely monitor the transformer's on-line behaviour. By monitoring of power transformers, it is possible to make the maximum practicable operating efficiency and optimum life of power transformers, minimise the risk of premature failures and provide the potential to change the maintenance strategy [1][2][3][4].

1.2 Main Methods of Fault Diagnosis for Power Transformer

There are a number of methods that can be used to monitor and diagnose power transformer faults, *e.g.* thermal modelling (TM), dissolved gas analysis (DGA), partial discharge analysis (PDA) and frequency response analysis (FRA), which will be briefly described in the following subsections.

1.2.1 Thermal modelling

The normal working life of a transformer is partially related to the deterioration of its insulation through thermal aging, which is determined mainly by its daily cyclic loadings. For oil immersed transformers, International Electrotechnical Commission (IEC) publication 60354 [5] can be used, while IEC publication 60905 [6] considers dry type transformers. In the Institute of Electrical and Electronics Engineers (IEEE) loading guide [7] the same basic calculation method and equations have been presented, as well as the standards from American National Standards Institute (ANSI) [8] and National Electrical Manufacturers Association (NEMA) [9].

Modelling transformer thermal dynamics is regarded as one of the most essential issues and the construction of an accurate thermal model is an important aspect of transformer condition monitoring. The generally accepted methods reported by IEC [5] and IEEE [7], can be used to predict the zones of hot-spot temperature in a transformer as the sum of the ambient temperature, mixed top-oil temperature rise above ambient and hot-spot rise above the mixed top-oil temperature. The two steady-state temperature rises of top-oil and bottom-oil above ambient can be estimated separately. The comparison between the evaluation of the calculated and the measured temperatures, which refer to the IEC power transformer thermal models, has been discussed in [1][4][10]. The TM is to use the data of Winding Temperature Indicator (WTI), Top-oil Temperature (TOT) and Bottom-oil Temperature (BOT) for transformer fault diagnosis. Table 1.1 shows the rules of the fault diagnosis using TM.

When WTI is beyond 75 degree, the cooler activates for dropping the temperature. When WTI is beneath 50 degree, the cooler shuts down and the cooling procedure aborted. When the temperature difference between TOT and BOT is more than 5 degree, the cooler will be activated. When WTI is more than 105 degree, the alarm signal is transmitted. When WTI keeps on increasing and reaches 120 degree, the triple alarm signal is activated for warning the users.

Table 1.1: The rules of TM

Temperature	Action
WTI > 75 degree	Cooler on
WTI < 50 degress	Cooler off
(TOT - BOT) > 5 degree	Cooler on
WTI > 105 degree	Alarm signal
WTI > 120 degree	Triple signal

1.2.2 Dissolved gas analysis

Power transformers are filled with a fluid that serves several purposes. The fluid acts as dielectric media, an insulator, and as a heat transfer agent. The most common type of fluid used in transformers is of a mineral oil origin. During normal use, there is usually a slow degradation of the mineral oil to yield certain gases that collect in the oil, *e.g.* C_2H_2, C_2H_4, CH_4, H_2 and C_2H_6. However, when there is an electrical fault within the transformer, gases are generated at a much more rapid rate. Dissolved gas analysis (DGA) is probably the most widely employed preventative maintenance technique in use today to monitor the operation of transformers [11][12][13][14]. The dissolved gases can be quantified and quantitatively determined by applying DGA interpretation techniques on an oil sample. The concentration and the relation of the individual gases allow a prediction of whether a fault has occurred or what type it is likely to be. For nearly 30 years, DGA and its interpretation had been a useful and reliable tool for the monitoring of the condition of oil-filled transformers and other oil-filled electrical equipment.

However, based on the conventional DGA interpretation methods, it is an arduous task to determine the malfunction types and the oil sampling intervals, due to the various faults' conditions and other interfering factors. Furthermore, determining the relationships between the gas levels and the decline conditions is a perplexing task, because of the complex gas combination patterns. Many attempts have been made to tackle the DGA problems with a few recent devel-

oped artificial intelligence (AI) techniques. Artificial neural networks (ANNs) are the most widely used faults classifier in DGA. In [15][16], an ANN is utilised to detect faults based only upon previous diagnostic results. Moreover, expert systems combined with other AI techniques, e.g. fuzzy models and evolutionary algorithms, have been developed for DGA [17][18]. These can evaluate the ongoing conditions and also suggest proper maintenance actions.

Table 1.2: Fault diagnosis table produced from IEC60559

Ratios of gases	$\frac{C_2H_2}{C_2H_4}$	$\frac{CH_4}{H_2}$	$\frac{C_2H_4}{C_2H_6}$
< 0.1	0	1	0
0.1 - 1	1	0	0
1 - 3	1	2	1
> 3	2	2	2

As a convenient basis for fault diagnosis, ratio methods are coding systems that assign a certain combination of codes to specific fault types. The codes are generated by calculating ratios of gas concentrations and comparing the ratios to pre-defined values, which have been derived from experience and are continually modified. A fault condition is detected when a code combination after calculation fits the code pattern of the fault. The most commonly used ratio method is Rogers Ratio Method [19], which is able to distinguish more types of thermal faults than the Dörnenberg Ratio Method.

First of all, definition and classification for the ratio of different gases are required. *eg.*, when the ratio of $\frac{C_2H_2}{C_2H_4}$ is greater than **0.1** and less than **1**. All the ratios of different gases are listed in Table 1.2. Under the circumstances that the ratio of $\frac{C_2H_2}{C_2H_4}$ is less than **0.1**, the ratio of $\frac{CH_4}{H_2}$ is more than **0.1** and less than **1**, the ratio of $\frac{C_2H_4}{C_2H_6}$ is less than **1**, there is no fault engenders, and the phenomena caused by ratio changes are listed in Table 1.3.

Table 1.3: Fault diagnosis table produced from IEC60559

Cases	Characteristic	$\frac{C_2H_2}{C_2H_4}$	$\frac{CH_4}{H_2}$	$\frac{C_2H_4}{C_2H_6}$
0	No fault	0	0	0
1	PDs of low energy density	0	1	0
2	PDs of high energy density	1	1	0
3	Discharge of low energy	1-2	0	1-2
4	Discharge of high energy	1	2	2
5	TF of low temperature<150 degree	0	0	1
6	TF of low temperature range 150-300 degree	0	2	0
7	TF of medium temperature range 300-700 degree	0	2	1
8	TF of high temperature>700 degree	0	2	2

1.2.3 Partial discharge analysis

Electrical insulation plays an important role in many high voltage power apparatuses, especially power transformers. Partial discharge (PD) occurs when the local electric field exceeds a threshold value, resulting in a partial breakdown of the surrounding medium as reported by IEC60270 [20]. Its cumulative effect leads to degradation of the insulation. PDs are initiated by the presence of defects in manufacture, or the choice of higher stress dictated by design considerations. Measurements are made to detect these PDs and monitor the soundness of insulation during the service life of the apparatus. PDs manifest as sharp current pulses at the terminals, whose nature depends on the type of insulation, defect present and measuring circuit, and detector used. The conventional electrical measurement of PDs is to detect the PD current pulses with a testing circuit. However, given that the experimental data we obtained consist of PD signals, sine waves, and background noise, the extracting of useful information from PD signals is a very important issue.

Detection of PDs may be performed by a variety of techniques, most commonly electric [4], acoustical [21], optical [22] and chemical techniques [23]. There are three types of analysis methods, the time-resolved partial discharge

analysis [23], the intensity spectra based PD analysis [24], and the phase resolved partial discharge analysis [25]. Because of the special characteristics of PDs, traditional digital signal processing methods are not suited for PD signal analysis. Other useful time-frequency tools, e.g. Fourier transform (FT) and Wavelet transform (WT), are used to analyse the PDs [26] for denoising, characteristic extraction and data classifications. On-line partial discharge calibration and monitoring for power transformers is also introduced with recently developed technology [27] [28].

1.2.4 Frequency response analysis

FRA is a comparative method for assessing the condition of power transformers. To evaluate the FRA results, actual data are compared with reference data either by direct visual inspection of the curves or by using processed FRA data. There are three approaches for comparing measured curves to reference data:

- Compare new measurements to fingerprint measurements on the same unit (time based comparison)
- Comparing measurements between identical (twin/sister) transformers (type based comparison)
- Comparing measurements made on symmetrical windings/limbs/phases on the same transformer (design based comparison)

When there is a reason to suspect mechanical damage (transport, extensive mechanical forces due to e.g. short circuit currents), the user can identify mechanical movements by comparing the reference curve with a curve obtained after the event. If the curves are identical, no internal displacements have occurred and the transformer can safely be put back to service. Individual measurements of every winding should be made which allows for identification of the location of the problem. For best results and most reliable analysis, a characteristic reference curve (fingerprint) of every winding should be captured

when the transformer is known to be in good condition. This profile is an investment as a future reference when the transformer has to be evaluated (time -based comparison).

Winding deformation may be due to mechanical or electrical faults. Mechanical faults occur for the reason of loss of properties, vibration throughout transportation as well as an inordinate mechanical force during a close-up short circuit fault, behave in the form of displaced winding, hoop buckling or damaged winding [29]. Winding movements may also due to stresses induced by electrical faults such as an inter turns short circuit owing to lightning strikes. These faults alter the distributed impedance parameters of the windings and therefore, can be detected by measuring the frequency response over a wide range of frequencies at the terminals of the transformer windings. FRA is an effective diagnostic method, which is more powerful than conventional measurements in detecting transformer winding deformation. It relies on the fact that transformer winding can be modelled as a network of capacitance, resistance, self-inductance and mutual inductance. When a fault occurs in the winding, the values of these parameters are altered and hence the frequency response from the winding will also change accordingly [29]. FRA is to use the frequency responses of winding in different frequency (*e.g.* low frequency (LF) is from 1kHz to 100kHz, medium frequency (MF) is from 100kHz to 600kHz, and high frequency (HF) is over 600kHz) for transformer faults diagnose. The diagnostic standard of frequency response analysis is shown in Table 1.4. R_{LF} stands for winding in low frequency, R_{MF} stands for winding in medium frequency and R_{HF} stands for winding in high frequency in Table 1.4. When R_{LF} is more than **1.0** and less than **2.0**, R_{MF} is more than **0.6** and less than **1.0**, R_{HF} is less than **0.6**, the winding results in mild deformation and the winding is required to surveillance. When R_{LF} is more than **0.6** and less than **1.0**, R_{MF} is less than **0.6**, the winding results in obvious deformation and the winding is required to maintenance. When R_{LF} is less than **0.6**, the winding results in serious deformation and the winding is required to replacement.

Table 1.4: Diagnostic standard of FRA

Winding deformation	R_{LF}	R_{MF}	R_{HF}	Suggustion
Serious deformation	R_{LF}<0.6			Replacement
Obvious deformation	1.0>R_{LF}>0.6	R_{MF}<0.6		Maintenance
Mild deformation	2.0>R_{LF}>1.0	0.6<R_{MF}<1.0	R_{HF}<0.6	Surveillance
No fault	R_{LF}>2.0	R_{MF}>1.0	R_{HF}>0.6	

1.3 Introduction to Ontologies and Web Ontology Language

1.3.1 Definitions

Historically, ontologies arise out of the branch of philosophy known as metaphysics, which deals with the nature of reality-of what exists. This fundamental branch is concerned with analysing various types or modes of existence, often with special attention to the relations between particulars and universals, between intrinsic and extrinsic properties, and between essence and existence. The traditional goal of ontological inquiry, in particular is to divide the world at its joints, to discover those fundamental categories, or kinds, into which the worlds objects naturally fall.

During the second half of the 20th century, philosophers extensively debated the possible methods or approaches to building ontologies, without actually building any very elaborate ontologies themselves. By contrast, computer scientists were building some large and robust ontologies with comparatively little debate over how they were built.

Since the mid-1970s, researchers in the field of artificial intelligence have recognised that capturing knowledge is the key to building large and powerful artificial intelligence (AI) systems. AI researchers argued that they could create new ontologies as computational models that enable certain kinds of automated reasoning. In the 1980s, the AI community began to use the term ontology

to refer to both a theory of a modelled world and a component of knowledge systems. Some researchers, drawing inspiration from philosophical ontologies, viewed computational ontology as a kind of applied philosophy.

In the early 1990s, the widely cited Web page and paper "Toward Principles for the Design of Ontologies Used for Knowledge Sharing" by Tom Gruber is credited with a deliberate definition of ontology as a technical term in computer science. Gruber introduced the term to mean a specification of a conceptualisation. That is, an ontology is a description, like a formal specification of a program, of the concepts and relationships that can exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set of concept definitions, but more general. And it is a different sense of the word than its use in philosophy.

Ontologies are often equated with taxonomic hierarchies of classes, class definitions, and the subsumption relation, but ontologies need not be limited to these forms. Ontologies are also not limited to conservative definitions-that is, definitions in the traditional logic sense that only introduce terminology and do not add any knowledge about the world. To specify a conceptualisation, one needs to state axioms that do constrain the possible interpretations for the defined terms.

In the early years of the 21st century, the interdisciplinary project of cognitive science has been bringing the two areas of scholars closer together. For example, there is talk of a computational turn in philosophy that includes philosophers analysing the formal ontologies of computer science (sometimes even working directly with the software), while researchers in computer science have been making more references to those philosophers who work on ontology (sometimes with direct consequences for their methods). Still, many scholars in both fields are uninvolved in this trend of cognitive science, and continue to work independently of one another, pursuing separately their different concerns.

Ontology

Ontology is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related. Web Ontologoy Language (OWL) is a language for defining and instantiating web ontologies. An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, *i.e.* facts not literally presented in an ontology, but entailed by the semantics. These entailments may be based upon a single document or multiple distributed documents in different knowledge bases, which have been combined using defined OWL mechanisms [30].

Ontologies are widely used to capture knowledge about some domains of interest. An ontology describes the concepts in a domain and also the relationships that hold between those concepts. The most recent development in standard ontology languages is OWL from the World Wide Web Consortium (W3C) [31]. It is based on a different logical model [32] which makes it possible for concepts to be defined as well as described. Complex concepts can be built up in definitions out of simpler concepts. Furthermore, the logical model allows the use of a reasoner which can check whether or not all the statements and definitions in an ontology are mutually consistent and can also recognize which concepts fit under which certain definitions [33]. The ontology model in Figure 1.1 includes four methods (TM, DGA, PDA and FRA) which can be used to monitor and diagnose power transformer faults.

Web ontology language

The OWL is a language for defining and instantiating Web ontologies. Ontology is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related. In addition, since 1990 there has evidently been another distinct strand to the research and development worked on software agents. Wooldridge and Jennings firstly proposed an intelligent agent concept which complemented and broadened the typology of agents being investigated by agent researchers [34]. Russell and

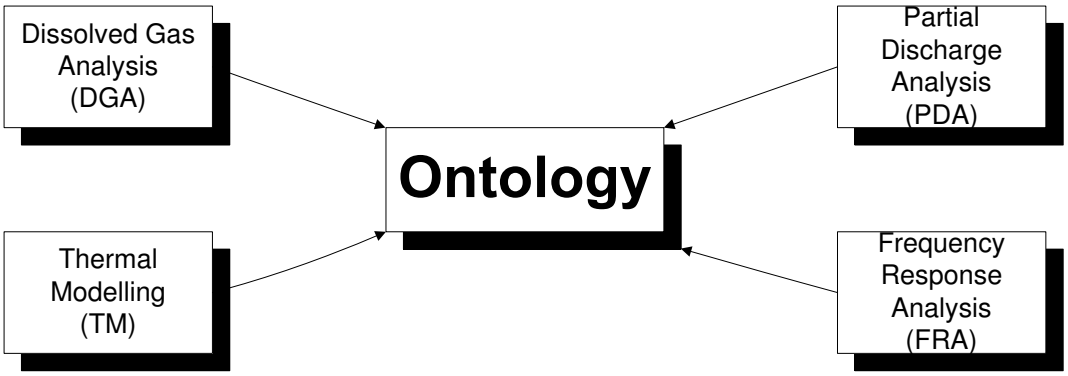

Figure 1.1: An ontology model for power transformer fault diagnosis.

Norvig extended an intelligent agent to a rational agent [35] that is to take actions based on information from and knowledge about the agent's environment. The rational agent tends to maximise the chances of success using commonly accepted logical inference rules.

One advantage of OWL ontologies is the availability of tools that can reason about them. Tools [31] can provide generic support that is not specific to a particular subject domain, which would be the case if one were to build a system to reason about a specific industry-standard XML Schema (XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes). Building a sound and useful reasoning system is not a simple effort [36].

Semantic web

The Semantic Web is an evolving extension of the World Wide Web in which the semantics of information and services on the web is defined, making it possible for the web to understand and satisfy the requests of people and machines to use the web content. It derives from W3C director Sir Tim Berners-Lee's vision of the Web as a universal medium for data, information, and knowledge exchange.

At its core, the semantic web comprises a set of design principles, collaborative working groups, and a variety of enabling technologies. Some elements

of the semantic web are expressed as prospective future possibilities that are yet to be implemented or realized.Other elements of the semantic web are expressed in formal specifications. Some of these include Resource Description Framework (RDF), a variety of data interchange formats, and notations such as RDF Schema (RDFS) and the OWL, all of which are intended to provide a formal description of concepts, terms and relationships within a given knowledge domain.

The Semantic Web is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web. The Semantic Web will build on XML's ability to define customized tagging schemes and RDF's flexible approach to representing data. The first level above RDF required for the Semantic Web is an ontology language what can formally describe the meaning of terminology used in Web documents. If machines are expected to perform useful reasoning tasks on these documents, the language must go beyond the basic semantics of RDFS. The OWL use cases and requirements document provides more details on ontologies, motivates the need for a OWL in terms of six use cases, and formulates design goals, requirements and objectives for OWL.

OWL has been designed to meet this need for a Web Ontology Language. OWL is part of the growing stack of W3C recommendations related to the Semantic Web [37].

- XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.
- XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes.
- RDF is a datamodel for objects ("resource") and relations between them, provides a simple semantics for this datamodel, and these datamodels can be represented in an XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of

RDF resources, with a semantics for generalization-hierarchies of such properties and classes.

- OWL adds more vocabulary for describing properties and classes: among others, relations between classes, cardinality, equality, richer typing of properties, characteristics of properties, and enumerated classes.

An ontology differs from an XML schema in that it is a knowledge representation, not a message format. Most industry based Web standards consist of a combination of message formats and protocol specifications.

1.3.2 The components of ontology

Contemporary ontologies share many structural similarities, regardless of the language in which they are expressed. As mentioned above, most ontologies describe individuals, classes, attributes, and relations. Figure 1.2 shows nine components in OWL ontology. Each of these components is discussed in below.

Common components of ontologies include:

- **Individuals**: instances or objects. Individuals (instances) are the basic, "ground level" components of an ontology. The individuals in an ontology may include concrete objects such as people, animals, tables, automobiles, molecules, and planets, as well as abstract individuals such as numbers and words (although there are differences of opinion as to whether numbers and words are classes or individuals). Strictly speaking, an ontology need not include any individuals, but one of the general purposes of an ontology is to provide a means of classifying individuals, even if those individuals are not explicitly part of the ontology.

- **Classes**: sets, collections, concepts, types of objects, or kinds of things. Classes concepts that are also called type, sort, category, and kind can be defined as an extension or an intension. According to an extensional definition, they are abstract groups, sets, or collections of objects. According to an intensional definition, they are abstract objects that are

defined by values of aspects that are constraints for being member of the class. The first definition of class results in ontologies in which a class is a subclass of collection. The second definition of class results in ontologies in which collections and classes are more fundamentally different. Classes may classify individuals, other classes, or a combination of both.

- **Attributes**: aspects, properties, features, characteristics. Objects in an ontology can be described by relating them to other things, typically aspects or parts. These related things are often called attributes, although they may be independent things. Each attribute can be a class or an individual. The kind of object and the kind of attribute determine the kind of relation between them. A relation between an object and an attribute express a fact that is specific to the object to which it is related.

- **Relations**: ways in which classes and individuals can be related to one another. Relationships (also known as relations) between objects in an ontology specify how objects are related to other objects. Typically a relation is of a particular type (or class) that specifies in what sense the object is related to the other object in the ontology. Much of the power of ontologies comes from the ability to describe relations. Together, the set of relations describes the semantics of the domain. The set of used relation types (classes of relations) and their subsumption hierarchy describe the expression power of the language in which the ontology is expressed.

- **Function terms**: complex structures formed from certain relations that can be used in place of an individual term in a statement.

- **Restrictions**: formally stated descriptions of what must be true in order for some assertion to be accepted as input.

- **Rules**: statements in the form of an if-then (antecedent-consequent) sentence that describe the logical inferences that can be drawn from an assertion in a particular form.

- **Axioms**: assertions (including rules) in a logical form that together comprise the overall theory that the ontology describes in its domain of application. This definition differs from that of "axiom" in generative grammar and formal logic. In these disciplines, axioms include only statements asserted as a priori knowledge. As used here,"axioms" also include the theory derived from axiomatic statements.

- **Events**: the changing of attributes or relations.

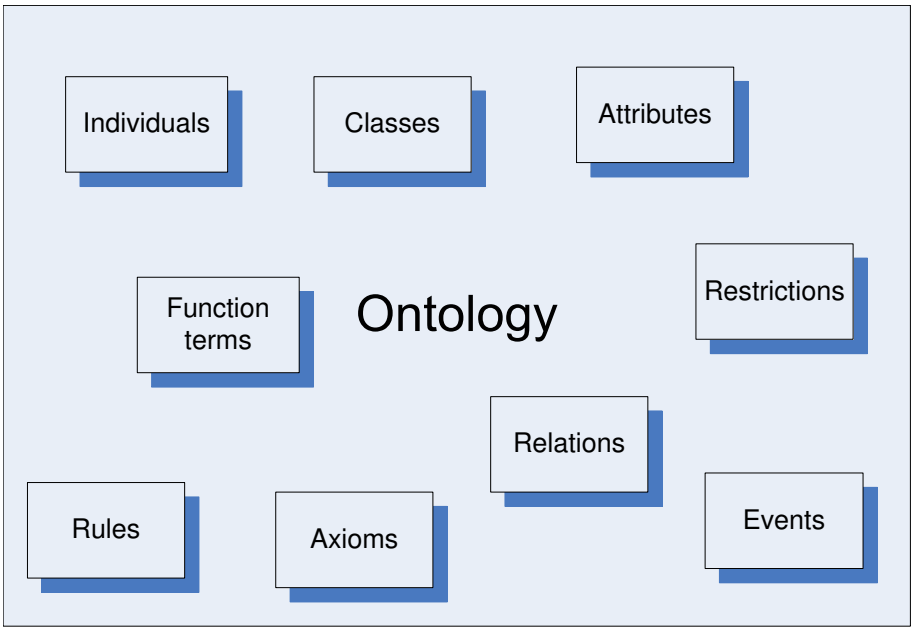

Figure 1.2: The components of the ontology.

1.3.3 Ontology languages

OWL may be categorised into three species or sub-languages: OWL-Lite, OWL-DL and OWL-Full. A defining feature of each sub-language is its expressiveness. OWL-Lite is the least expressive sub-language. OWL-Full is the most expressive sub-language. The expressiveness of OWL-DL falls between that of OWL-Lite and OWL-Full. OWL-DL may be considered as an extension of OWL-Lite and OWL-Full an extension of OWL-DL.

- OWL-Lite is the syntactically simplest sub-language. It is intended to be used in situations where only a simple class hierarchy and simple constraints are needed. For example, it is envisaged that OWL-Lite will provide a quick migration path for existing thesauri and other conceptually simple hierarchies. OWL-Lite supports those users primarily needing a classification hierarchy and simple constraint features. For example, while OWL-Lite supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL-Lite than its more expressive relatives, and provide a quick migration path for thesauri and other taxonomies. Owl-Lite also has a lower formal complexity than OWL-DL.

- OWL-DL is much more expressive than OWL-Lite. OWL-DL and OWL-Lite are based on Description Logics (hence the suffix DL). Description Logics are a decidable fragment of First Order Logic and are therefore amenable to automated reasoning. It is therefore possible to automatically compute the classification hierarchy and check for inconsistencies in an ontology that conforms to OWL-DL. OWL-DL supports those users who want the maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) of reasoning systems. OWL-DL includes all OWL language constructs with restrictions such as type separation (a class can not also be an individual or property, a property can not also be an individual or class). OWL-DL is so named due to its correspondence with description logics (DL), a field of research that has studied a particular decidable fragment of first order logic. OWL-DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems.

- OWL-Full is the most expressive OWL sub-language. It is intended to be used in situations where very high expressiveness is more important

than being able to guarantee the decidability or computational completeness of the language. It is therefore not possible to perform automated reasoning on OWL-Full ontologies. OWL-Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL-Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. Another significant difference from OWL-DL is that a `owl:DatatypeProperty` can be marked as a `owl:InverseFunctionalProperty`. OWL-Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support every feature of OWL-Full.

Each of these sub-languages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded. The following set of relations hold. Their inverses do not.

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

1.3.4 Choosing the sub-language

On adopting OWL, the ontology developers should consider which language of OWL is the most suitable one for their needs. The choice between OWL-Lite and OWL-DL depends on the extent to which users require the more expressive restriction constructs provided by OWL-DL. Reasoners for OWL-Lite will have desirable computational properties. Reasoners for OWL-DL, while dealing with a decidable sub-language, will be subject to the higher worst-case complexity. The choice between OWL-DL and OWL-Full mainly

depends on the extent to which users require the meta-modelling facilities of RDF Schema (*i.e.* defining classes of classes). While using OWL-Full in comparison with OWL-DL, reasoning support is less predictable since complete OWL-Full implementations do not currently exist.

OWL-Full can be viewed as an extension of RDF, while OWL-Lite and OWL-DL can be viewed as extensions of a restricted view of RDF. Every OWL (Lite, DL, Full) document is an RDF document, and every RDF document is an OWL-Full document, but only some RDF documents will be a legal OWL-Lite or OWL-DL document. Because of this, some care has to be taken when a user wants to migrate from an RDF document to OWL. When the expressiveness of OWL-DL or OWL-Lite is deemed appropriate, some precautions have to be taken to ensure that the original RDF document complies with the additional constraints imposed by OWL-DL and OWL-Lite. Among others, every URI that is used as a class name must be explicitly asserted to be of type `owl:Class` (and similarly for properties), every individual must be asserted to belong to at least one class (even if only `owl:Thing`), the URI's used for classes, properties and individuals must be mutually disjoint.

Although many factors should be taken into account on deciding the appropriate sub-language to use, there are some simple rules of thumb [37]. User can follow the following two rules.

- The choice between OWL-Lite and OWL-DL may be based upon whether the simple constructs of OWL-Lite are sufficient or not.
- The choice between OWL-DL and OWL-Full may be based upon whether it is important to be able to carry out automated reasoning on the ontology or whether it is important to be able to use highly expressive and powerful modelling facilities such as meta-classes (classes of classes).

Individuals, represent objects in the domain in which we are interested. OWL does not use the Unique Name Assumption (UNA). This means that two different names could actually refer to the same individual. For example, Electrical Engineering and Electronics and EEE might all refer to the same

individual. In OWL, it must be explicitly stated that individuals are the same as each other, or different to each other, otherwise they might be the same as each other, or they might be different to each other. Properties are binary relations on individuals. *i.e.* properties link two individuals together. For example, the property *is a part of* links the individual **School of Engineering** to the individual **University of Liverpool**, and the property *is a part of* also links the individual **Electrical Engineering and Electronics** to the individual **School of Engineering**. They can also be either transitive or symmetric. OWL classes are interpreted as sets that contain individuals. They are described using formal descriptions that state precisely the requirements for membership of the class. For example, the class **Department** would contain all the individuals (**Electrical Engineering and Electronics** and **Computer Sciences** *etc.*). One of the key features of OWL-DL is that these superclass-subclass relationships can be computed automatically by a reasoner. Figure 1.3 shows a representation of some individuals in some domain, some properties linking some individuals together and some classes containing individuals.

1.3.5 Expert-system

Conventional approaches for power system maintenances are carried out by a so called Expert-System (ES), formed of experts from different disciplines. However, with hundreds of substations in a modern power system and developments in data acquisition and control devices, much more information is being gathered and stored. This increases the size of the available knowledge base, and hence should enable more efficient operation of the plant, and assist in maintenance and other activities. However, the amount of data available is much greater than these can be effectively used by human engineers. Therefore, new techniques for automated data analysis and decision making are required, in order to filter the amount of data provided to engineers and reduce the need for human intervention in simpler situations, which could be used automatically.

In artificial intelligence, an expert system is a computer system that em-

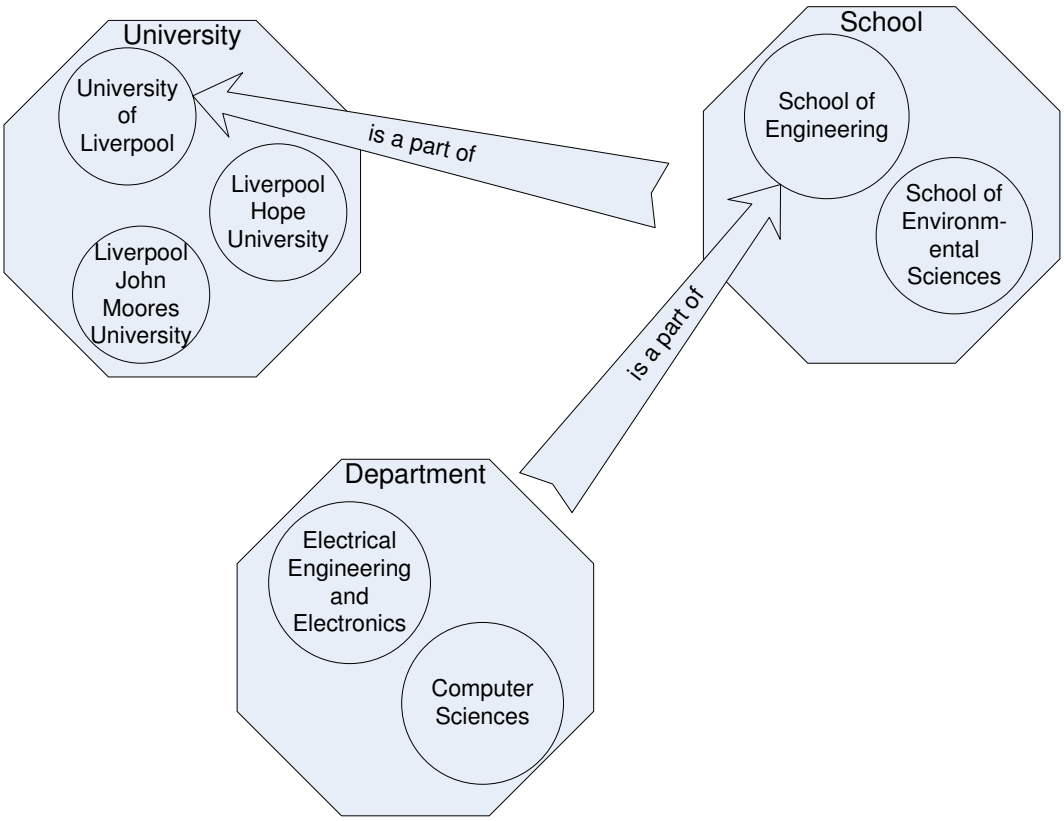

Figure 1.3: Representation of individuals, properties and classes.

ulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning about knowledge. An expert system has a unique structure, different from traditional programs. It is divided into two parts, one fixed, independent of the expert system: the inference engine, and one variable: the knowledge base [38]. To run an expert system, the engine reasons about the knowledge base like a human. In the 80's a third part appeared: a dialog interface to communicate with users [39]. This ability to conduct a conversation with users was later called "conversational". The inference engine is a computer program designed to produce a reasoning on rules. In order to produce a reasoning, it is based on logic. There are several kinds of logic: propositional logic, predicates of order 1 or more, epistemic logic, modal logic, temporal logic, fuzzy logic, *etc.* Except propositional logic, all are com-

plex and can only be understood by mathematicians, logicians or computer scientists. Propositional logic is the basic human logic, that expressed in the syllogism. The expert system that uses that logic are also called zeroth-order expert system. With logic, the engine is able to generate new information from the knowledge contained in the rule base and data to be processed.

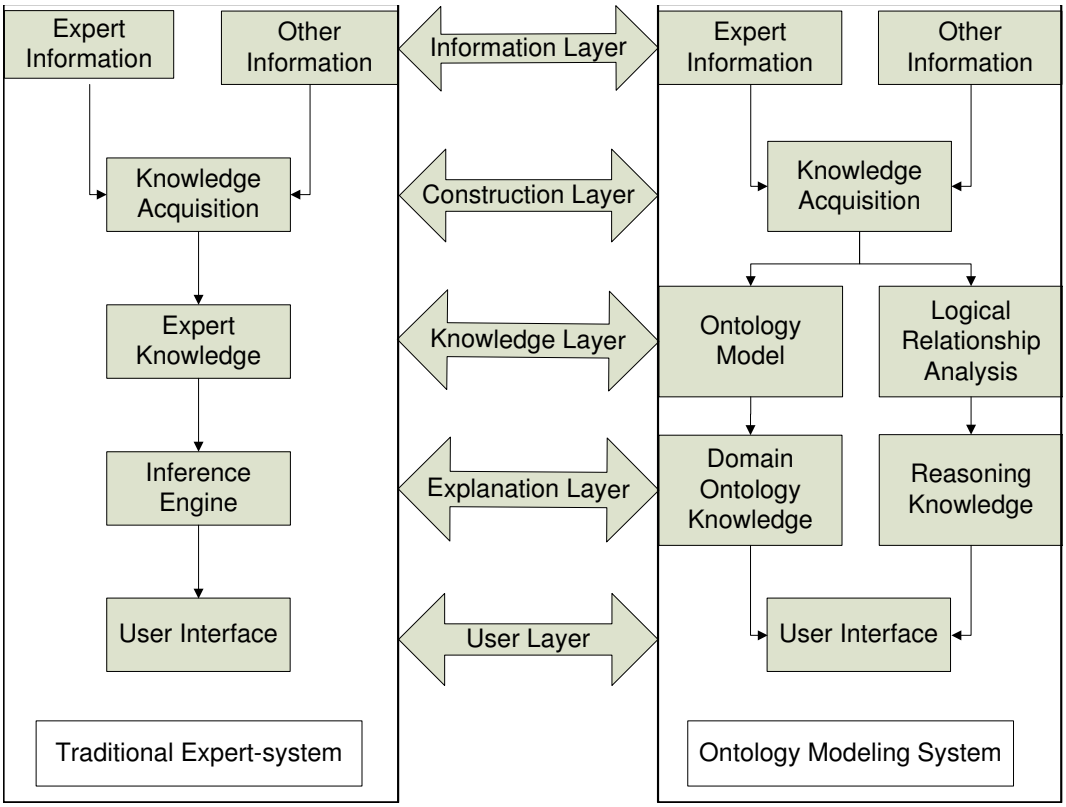

Figure 1.4: Comparison between traditional expert-system and ontology modeling system.

The traditional ES is provided with strong pertinence yet the expansibility and study function of a knowledge base is comparatively weak. As a result of the diversified inference mechanism and knowledge library structure, the former problems on knowledge exchange in many systems could never be solved. So the purpose of introducing ontology in AI is precisely for enhancing the exchanging-study ability between heterogeneous systems as well as realising authentic knowledge exchange. Figure 1.4 is a comparison between a tradi-

tional ES, and ontology based system. As the knowledge foundation of the whole system, an ontology knowledge library guarantees the realisation of a higher-level knowledge exchange.

1.4 Thesis Outline

This thesis is structured as follows:

Chapter 2 firstly introduced how to develop ontology programs with OWL in seven-steps. Then inducted expert system and compared the difference between traditional expert system and the ontology-based diagnosis system. The analysis of fault diagnosis system structure explained the relations between different part and elaborated the definition and application of each component of ontology. Finally, with a detailed explanation of OWL, a primary ontology model for power transformer fault diagnosis was established based on TM.

Chapter 3 firstly introduced Protégé, the software used to build an ontology model. Then an ontology model was built with Protégé based on frequency response analysis. After developed and supplemented of this model, it was integrated with TM, DGA, PDA and FRA of ontology by which it extended to be query system aims at power transformer fault diagnosis. Finally, based on the ontology model, an interface for the ontology-based power transformer fault diagnosis system was built up by Java.

Chapter 4 concludes the thesis based on the outcomes obtained in this study, followed by the discussion of the challenges of this work. Suggestions for future work are also listed in this chapter.

1.5 Autobibliography

List of the publication produced from this work:

1. **Wang, D.**, Tang, W. H. and Wu, Q. H. Ontology-based fault diagnosis for power transformers. In *Power and Energy Society General Meeting, 2010 IEEE*. July 25–29, 2010.

Chapter 2

An Ontology Model for Transformer Fault Diagnosis

In this chapter, an ontology model for transformer fault diagnosis is built by protégé. Before the construction of our ontology base, to be clarified, the ontology base must be placed in fault diagnosis, which is an extroverted application area, hence ontology will pay more attention to class and the relationship between classed, and place the private attribute of class and other microcosmic concept in the secondary position.

2.1 Introduction

Generally speaking, ontology provides a group of terminology and concepts to describe one certain domain and the knowledge-base uses these terminologies to express the facts of this domain. For example, the ontology of a power transformer fault diagnosis system may possibly contain high frequency, mild deformation, maintenance or other terminologies. However, it will not contain the recommend contract from some fault situations to some fault sources, or even to repair methods.

Seven-steps method

The AI literature contains many definitions of an ontology; many of these

contradict from others. An ontology is a formal explicit description of concepts in a domain of discourse (classes (sometimes called concepts)), properties of each concept describing various features and attributes of the concept (slots (sometimes called roles or properties)), and restrictions on slots (facets (sometimes called role restrictions)). An ontology together with a set of individual instances of classes constitutes a knowledge base. In reality, there is a fine line where the ontology ends and the knowledge base begins. The best solution almost always depends on the application that users have in mind and the extensions that users anticipate. Ontology development is necessarily an iterative process. Concepts in the ontology should be close to objects (physical or logical) and relationships in domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe domain. An ontology is a model of reality of the world and the concepts in the ontology must reflect this reality. After defining an initial version of the ontology, user can evaluate and debug it by using it in either applications or problem-solving methods or by discussing it with experts in the field, or both. As a result, revising the initial ontology becomes a must. This process of iterative design will likely be continued through the entire life cycle of the ontology.

The most common methods of designing an ontology is the ***Seven-Step method*** [40].

- **Step 1. Determine the domain and scope of the ontology**
  One of the methods to determine the scope of the ontology is to sketch a list of questions that a knowledge base based on the ontology should be able to answer questions.

- **Step 2. Consider reusing existing ontologies**
  It is almost always worth considering what someone else has done and checking if the existing sources for particular domain and task can be refined and extended. Reusing existing ontologies may be a requirement if our system needs to interact with other applications that have already committed to particular ontologies or controlled vocabularies. Many ontologies are already available in the electronic form and can be imported

into an ontology-development environment. The formalism in which an ontology is expressed often does not matter, since many knowledge-representation systems can import and export ontologies. Even if a knowledge-representation system cannot work directly with a particular formalism, the task of translating an ontology from one formalism to another is usually not a difficult one. There are libraries of reusable ontologies on the Web and in the literature.

- **Step 3. Enumerate important terms in the ontology**
  Initially, it is important to get a comprehensive list of terms without worrying about overlap between concepts they represent, relations among the terms, or any properties that the concepts may have, or whether the concepts are classes or slots.. The next two steps developing the class hierarchy and defining properties of concepts (slots) are closely intertwined. It is hard to do one of them first and then do the other. Typically, we create a few definitions of the concepts in the hierarchy and then continue by describing properties of these concepts and so on. These two steps are also the most important steps in the ontology-design process.

- **Step 4.** Define the classes and the class hierarchy
  There are several possible approaches in developing a class hierarchy:

  – **Top-down** development process starts with the definition of the most general concepts in the domain and subsequent specialisation of the concepts as shown in Figure 2.1.

  – **Bottom-up** development process starts with the definition of the most specific classes, the leaves of the hierarchy, with subsequent grouping of these classes into more general concepts.

  – **Combination** development process is a combination of the top-down and bottom-up approaches: define the more salient concepts first and then generalise and specialise them appropriately.

On the realization of the ontology construction, top-down method is rec-

ommended to be applied for its three advantages. Firstly, top-down method can ensure the ontology to be well-established and comprehensive. Furthermore, top-down method is more acceptable because it is accordant with peoples logic of describing objects in daily life. Finally, it is more convenient to supplement and modify after the construction by using the top-down method.

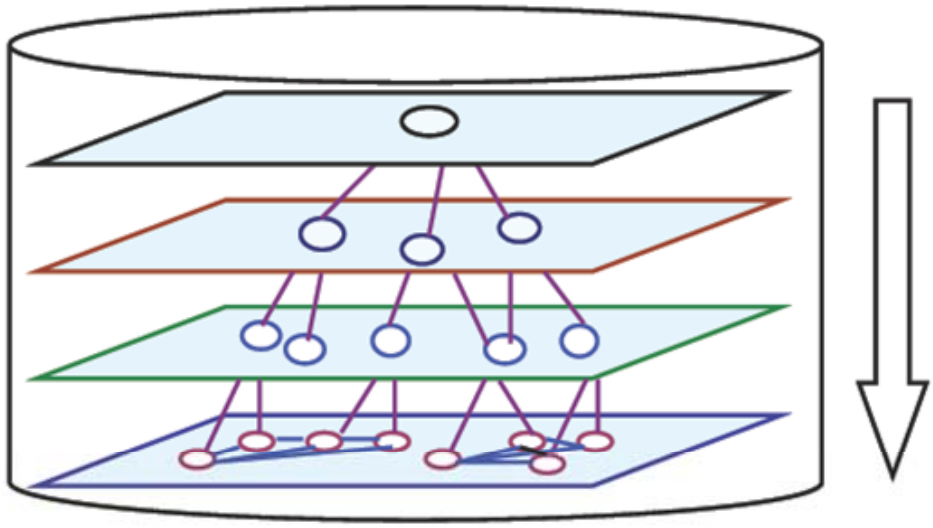

Figure 2.1: Class definition in Top-down method.

- **Step 5.** Define the properties of classes-slots
  In general, there are several types of object properties that can become slots in an ontology:
  - intrinsic properties ;
  - extrinsic properties ;
  - parts, if the object is structured; these can be both physical and abstract parts
  - relationships to other individuals; these are the relationships between individual members of the class and other items

- **Step 6.** Define the facets of the slots
  Slots can have different facets describing the value type, allowed values, the number of the values (cardinality), and other features of the values the slot can take.

- **Step 7.** Create instances
  The last step is creating individual instances of classes in the hierarchy. Defining an individual instance of a class requires choosing a class,creating an individual instance of that class, and filling in the slot values.

Generally speaking, the key point of building ontology is to learn the structure of the ontology, establish it from the top level to the bottom level and define each level with an explicit description to make the ontology comprehensive and all-inclusive. A well-established ontology can ensure the efficient and quick query on the content of ontology.

2.2 The Structure of Fault Diagnosis System Based on Ontology

The basic structure of the proposed fault diagnosis system is generally composed of three parts: fault phenomenon, fault reason and fault source, and many fault diagnosis systems are provided with the repair suggestions as well. Based on the above understanding and the request of the system, the following definitions are given.

2.2.1 The structure of the fault diagnosis system

The ontology process of transformer fault diagnosis is the same as the description of the process of diagnosis. It starts from a fault phenomenon (Phn), points to a fault reason (Rsn), terminates at a repair suggestion (Rpr) and finds a fault source (Sou). Equation 1 illustrates the "Relational model" of the system includeing four elements.

$$\mathrm{FD} =< \mathrm{Phn}, \mathrm{Rsn}, \mathrm{Sou}, \mathrm{Rpr} > \tag{2.2.1}$$

Fault phenomenon non-empty finite sets are:

$$\mathrm{Phn} = \{\mathrm{Phn1}, \mathrm{Phn2}, ..., \mathrm{PhnL}\} \tag{2.2.2}$$

Fault reason non-empty finite sets are:

$$\mathrm{Rsn} = \{\mathrm{Rsn1}, \mathrm{Rsn2}, ..., \mathrm{RsnL}\} \tag{2.2.3}$$

Repair suggestion non-empty finite sets are:

$$\mathrm{Rpr} = \{\mathrm{Rpr1}, \mathrm{Rpr2}, ..., \mathrm{RprL}\} \tag{2.2.4}$$

Fault source non-empty finite sets are:

$$\mathrm{Sou} = \{\mathrm{Sou1}, \mathrm{Sou2}, ..., \mathrm{SouL}\} \tag{2.2.5}$$

Obviously, in the definition process of the fault diagnosis system, there is no such a key word, similar to the key word "subject". There are so many similar key words in the fault diagnosis system so that the entire diagnosis process does not contain only one match retrieval. Just like the definition expresses, the aim of a power transformer fault diagnosis system is basically to find the most possible link path which could connect fault phenomena and fault sources, even repair suggestions through logical reasoning. Meanwhile, it is obvious that the frame of the system is constructed when the basic ontology in this domain and its relations are defined. Since the basic structure of a fault diagnosis system is generally composed of four parts: fault phenomenon, fault reason, repair suggestion and fault source.

There are thirteen foundational relations, which should be initially defined in the ontology knowledge base:

$$as\ result\ of : Fault\ Phenomenon -> Fault\ Reason \tag{2.2.6}$$

$$result\ in : Fault\ Reason -> Fault\ Phenomenon \tag{2.2.7}$$

$$reason\ to\ repair : Fault\ Reason -> Repair\ Suggestion \tag{2.2.8}$$

$$repair\ to\ reason : Repair\ Suggestion -> Fault\ Reason \tag{2.2.9}$$

$$repairing : Repair\ Suggeation -> Fault\ Source \tag{2.2.10}$$

$$for\ repair : Fault\ Source -> Repair\ Suggeation \tag{2.2.11}$$

$$have\ phenomenon : Fault\ Source -> Fault\ Phenomenon \tag{2.2.12}$$

$$phenomenon\ on : Fault\ Phenomenon -> Fault\ Source \tag{2.2.13}$$

$$to\ solve : Repair\ Suggeation -> Fault\ Phenomenon \tag{2.2.14}$$

$$solve\ via : Fault\ Phenomenon -> Repair\ Suggeation \tag{2.2.15}$$

$$affairs\ on : Fault\ Reason -> Fault\ Source \tag{2.2.16}$$

$$have\ affair : Fault\ Source -> Fault\ Reason \tag{2.2.17}$$

$$is\ a\ part\ of : Subclass -> Class \tag{2.2.18}$$

Among these relations, *is a part of* is a special relation. On one hand, it represents the Inclusion relations between class and class, or between class and

individual. On the other hand, it is a transitive relation. For example, class A is the subclass of class B. Class B is the subclass of Class C. So class A is the subclass of Class C. In other words, a class is the subclass of its upper-leveled class. At the same time, this class is the subclass of a much upper-leveled class, or the subclass of the top class. The property of "is a part of" represents the transitive property.

The thirteen relations contain all concepts in the whole system and all the related concepts in the system have already been described and connected through the thirteen basic relational operators. At the same time, according to the ontology relation structure, all the other relations in the system are inferential through the twelve relations. Moreover, in the ontological knowledge base, the definition of this inference could be momentarily expanded by adding inference knowledge into the inference base.

2.2.2 Namespaces

A standard initial component of an ontology includes a set of XML namespace declarations enclosed in an opening `rdf:RDF` tag. These provide a means to unambiguously interpret identifiers and make the rest of the ontology presentation much more readable. A typical OWL ontology begins with a namespace declaration similar to the following.

```
<rdf:RDF
xmlns="http://www.diagnosis.com/ontologies/diagnosis.owl#"

xmlns:diagnosis="http://www.diagnosis.com/ontologies/diagnosis.owl#"

xml:base="http://www.diagnosis.com/ontologies/diagnosis.owl#"

xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"

xmlns:owl="http://www.w3.org/2002/07/owl#"
```

```
xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#

xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#

xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
```

The first two declarations identify the namespace associated with this ontology. The first makes it the default namespace, stating that unprefixed qualified names refer to the current ontology. The second identifies the namespace of the current ontology with the prefix `diagnosis`. The third identifies the base URI (Uniform Resource Identifier) for this document. The fourth identifies the namespace of the supporting owl2xml ontology with the prefix `owl2xml`. The fifth namespace declaration says that in this document, elements prefixed with `owl:` should be understood as referring to things drawn from the namespace called `http://www.w3.org/2002/07/owl#`. This is a conventional OWL declaration, used to introduce the OWL vocabulary. OWL depends on constructs defined by RDF, RDFS, and XML Schema datatypes. In this document, the `rdf:` prefix refers to things drawn from the namespace called `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. The next two namespace declarations make similar statements about the RDF Schema `rdfs:` and XML Schema datatype `xsd:` namespaces.

2.2.3 Ontology headers and data aggregation

Once namespaces are established, normally include a collection of assertions about the ontology grouped under an `owl:Ontology` tag. These tags support such critical housekeeping tasks as comments, version control and inclusion of other ontologies.

```
<owl:Ontology rdf:about="">

<rdfs:comment>
```

```
An example OWL ontology model for power transformer fault diagnosis
</rdfs:comment>

<owl:priorVersion rdf:resource=
"http://www.diagnosis.com/ontologies/diagnosis.owl#"/>

<owl:imports rdf:resource="http://www.w3.org/2006/12/owl2-xml#"/>

<rdfs:label>Diagnosis Ontology

</rdfs:label>
...
```

- The `owl:Ontology` element is a place to collect plenty of OWL meta-data for the document. It does not guarantee that the document describes an ontology in the traditional sense. In some communities, ontologies are not about individuals but only the classes and properties that define a domain. When using OWL to describe a collection of instance data the `owl:Ontology` tag may be needed in order to record version information and to import the definitions that the document depends on. Thus, in OWL the term ontology has been broadened to include instance data.

- The `rdf:about` attribute provides a name or reference for the ontology. The standard case, the name of the ontology is the base URI of the `owl:Ontology` element. Typically, this is the URI of the document containing the ontology. An exception to this is a context that makes use of `xml:base` which may set the base URI for an element to something other than the URI of the current document.

- `rdfs:comment` provides the obviouly needed capability to annotate an ontology.

- `owl:priorVersion` is a standard tag intended to provide hooks for version control systems working with ontologies.

- `owl:imports` provides an include-style mechanism. `owl:imports` takes a single argument, identified by the `rdf:resource` attribute.

Importing another ontology brings the entire set of assertions provided by that ontology into the current ontology. In order to make best use of this imported ontology it would normally be coordinated with a namespace declaration. Notice the distinction between these two mechanisms. The namespace declarations provide a convenient means to reference names defined in other OWL ontologies. Conceptually, `owl:imports` is provided to indicate intention to include the assertions of the target ontology.

OWL provides several other mechanisms to tie the current ontology and imported ontologies together. Also include a `rdfs:label` to support a natural language label for ontology.

The ontology header definition is closed with the following tag.

```
</owl:Ontology>
```

This prelude is followed by the actual definitions that make up the ontology and is ultimately closed by

```
</rdf:RDF>
```

OWL ability to express ontological information about instances appearing in multiple documents supports linking of data from diverse sources in a principled way. The underlying semantics provides support for inferences over this data that may yield unexpected results. In particular, the ability to express equivalences using `owl:sameAs` can be used to state that seemingly different individuals are actually the same.

2.2.4 Classes and individuals

The most basic concepts in a domain should correspond to classes that are the roots of various taxonomic trees. Every individual in the OWL world is a member of the class `owl:Thing`. Thus each user-defined class is implicitly

a subclass of `owl:Thing`. Domain specific root classes are defined by simply declaring a named class. OWL also defines the empty class, `owl:Nothing`.

```
<owl:Class rdf:ID="FaultPhenomenon"/>

<owl:Class rdf:ID="FaultReason"/>

<owl:Class rdf:ID="FaultSource"/>

<owl:Class rdf:ID="RepairSuggestion"/>
```

The syntax `rdf:ID="FaultPhenomenon"` is used to introduce a name, as part of its definition. This is the rdf:ID attribute RDF that is like the familiar ID attribute defined by XML.

The fundamental taxonomic constructor for classes is `rdfs:subClassOf`. It relates a more specific class to a more general class. If **X** is a subclass of **Y**, then every instance of **X** is also an instance of **Y**. The `rdfs:subClassOf` relation is transitive. If **X** is a subclass of **Y** and **Y** a subclass of **Z** then **X** is a subclass of **Z**.

```
<owl:Class rdf:ID="Rlf_is_Bigger_than_2.0"><rdfs:subClassOf>

<owl:Class rdf:ID="Value_of_Winding_Resistance_in_Low_Frequency"/>

</rdfs:subClassOf>

</owl:Class>
```

The `Rlf_is_Bigger_than_2.0` is defined to be a subclass of `Value_of_Winding_Resistance_in_Low_Frequency`.

A class definition has two parts: a name introduction or reference and a list of restrictions. Each of the immediate contained expressions in the class definition further restricts the instances of the defined class. Instances of the

class belong to the intersection of the restrictions. At this point it is possible to create a simple (and incomplete) definition for the class Diagnosis. `Value_of_Ratio_of_CH4_and_H2` is a `Fault_Reason`. Also define `Cooler_On` as an `Repair_Suggestion`.

```
<owl:Class rdf:ID="#Value_of_Ratio_of_CH4_and_H2">

<rdfs:subClassOf rdf:resource="Fault_Reason"/>

...

</owl:Class>
```

and

```
<owl:Class rdf:ID="Cooler_On">

<rdfs:subClassOf rdf:resource="#Repair_Suggestion" />

...

</owl:Class>
```

In addition to classes, user need to be able to describe class members. Normally thinking of these as individuals in the universe of things. An individual is minimally introduced by declaring it to be a member of a class.

```
<Value_of_Ratio_of_C2H2_and_C2H4 rdf:ID="ratio_one_is_bigger_than_1">
```

`rdf:type` is an RDF property that ties an individual to a class of which it is a member.

There are a couple points of view be made here. First, deciding that `ratio_one_is_bigger_than_1` is member of `Value_of_Ratio_of_C2H2_and_C2H4`,

the class containing all value of ratio of C_2H_2 and C_2H_4. Second, there is no requirement in the two-part example that the two elements need to be adjacent to one another, or even in the same file (though the names would need to be extended with a URI in such a case).

There are important issues regarding the distinction between a class and an individual in OWL. A class is simply a name and collection of properties that describe a set of individuals. Individuals are the members of those sets. Thus classes should correspond to naturally occurring sets of things in a domain of discourse, and individuals should correspond to actual entities that can be grouped into these classes.

In building ontologies, this distinction is frequently blurred in two ways:

- **Levels of representation**: In certain contexts something that is obviously a class can itself be considered an instance of something else. For example, in the diagnosis ontology, the ontology has the notion of a `Fault_Phenomenon`, which is intended to denote the set of all fault phenomena. `Obvious_Deformation` is an example instance of this class, as it denotes the actual `Obvious_Deformation` is one instance of the fault phenomenon. However, `Obvious_Deformation` could itself be considered a class, the set of all actual obvious fault deformations.

- **Subclass and instance**: It is most likely to confuse the instance-of relationship with the `subclass` relationship. For example, it may seem arbitrary to choose to make `Obvious_Deformation` an individual that is an instance of `Fault_Phenomenon`, as opposed to a subclass of `Fault_Phenomenon`. This is not an arbitrary decision. The `Fault_Phenomenon` class denotes the set of all fault phenomena, and therefore any subclass of `Fault_Phenomenon` should denote a subset of these faults. Thus, `Obvious_Deformation` should be considered an instance of `Fault_Phenomenon`, and not a subclass.

The point of this discussion is to note that the development of an ontology should be firmly driven by the intended usage. These issues also underlie one

major difference between OWL-Full and OWL-DL. OWL-Full allows the use of classes as instances and OWL-DL does not. The diagnosis ontology is designed to work in OWL-DL, and as a result individuals are not simultaneously treated as classes.

2.2.5 Defining properties and datatypes

The property is a binary relation. Two types of properties are distinguished: `datatype properties`: relations between instances of classes and RDF literals and XML Schema datatypes. And `object properties`: relations between instances of two classes. When user defines a property there are a number of ways to restrict the relation. The domain and range can be specified. The property can be defined to be a specialization of an existing property.

```
<owl:ObjectProperty rdf:about="#solve_via">

<rdfs:range rdf:resource="#Repair_Suggestion"/>

<rdfs:domain rdf:resource="#Fault_Phenomenon"/>

</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#as_result_of">

<rdfs:range rdf:resource="#Fault_Reason"/>

<rdfs:domain rdf:resource="#Fault_Phenomenon"/>

</owl:ObjectProperty>
```

In OWL, a sequence of elements without an explicit operator represents an implicit conjunction. The property `solve_via` has a domain of `Fault_Phenomenon` and a range of `Repair_Suggestion`. It relates instances of the class `Repair_Suggestion`

to instances of the class `Fault_Phenomenon`. Multiple domains mean that the domain of the property is the intersection of the identified classes.

Similarly, the property course ties a `Fault_Reason` to a `Fault_Phenomenon`. Distinguishing properties according to whether they relate individuals to individuals (object properties) or individuals to datatypes (datatype properties). Datatype properties may range over RDF literals or simple types defined in accordance with XML Schema datatypes.

2.2.6 Property characteristics

If a property, **P**, is specified as transitive then for any **x**, **y**, and **z**: `P(x,y)` and `P(y,z)` implies `P(x,z)`. The property `is_part_of` is transitive.

```
<owl:ObjectProperty rdf:ID="is_part_of">

<rdf:type rdf:resource="&owl;TransitiveProperty" />

<rdfs:domain rdf:resource="&owl;Thing" />

<rdfs:range rdf:resource="#Fault_Phenomenon" />

</owl:ObjectProperty>

<Fault_Phenomenon rdf:ID="WTI_is_Bigger_Than_105C">

<locatedIn rdf:resource="#Value_of_WTI" />

</Region>
```

Because the `WTI_is_Bigger_Than_105C` `is_part_of` the `Value_of_WTI`, then it must also `is_part_of` the `Fault_Phenomenon`, since `is_part_of` is transitive.

If a property **P1** is tagged as the `owl:inverseOf` **P2**, then for all **x** and **y**:

`P1(x,y)` iff `P2(y,x)`

Note that the syntax for `owl:inverseOf` takes a property name as an argument. **A** iff **B** means (**A** implies **B**) and (**B** implies **A**).

```
<owl:inverseOf>

<owl:ObjectProperty rdf:ID="repair_via"/>

</owl:inverseOf>

<rdfs:domain rdf:resource="#Repair_Suggestion"/>

<rdfs:range rdf:resource="#Fault_Structure"/>

</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="phenomenon_on">

<owl:inverseOf>

<owl:ObjectProperty rdf:ID="has_phenomenon"/>

</owl:inverseOf>
```

That means the `phenomenon_on` and `has_phenomenon` are reversible.

Chapter 3

Query System of Power Transformer Fault Diagnosis

In this thesis, the diagnosis techniques of thermal modeling, dissolved gas analysis, partial discharge analysis and frequency response analysis are integrated into an ontology model using by OWL. To ensure the user a simple and fast operation on the transformer fault diagnosis and examination, a transformer diagnosis model based on ontology is required to be established. Therefore, a well-established OWL editing software should be used on the description of the above-mentioned ontology model so as to enable the non-professionals to use this model conveniently. This chapter will introduce how to use a proper software to describe the ontology model and establish a transformer diagnosis system that could be used by non-professionals. An ontology modeling system for power transformer fault diagnosis is built by Protégé.

3.1 Softwares Introduction

To achieve the research aim of this thesis, four softwares are introduced in this thesis.

- **Protégé**: Software of establishing ontology
- **Graphviz**: Software of generating ontology tree graphs

- **Java**: Language program software on building the interface.
- **Eclipse**: Editing and developing software for Java program.

3.1.1 Protégé

Protégé is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modelling structures and actions that support the creation, visualisation, and control of ontologies in various representation formats. Protégé can be customised to provide domain-friendly support for creating knowledge models and entering data. Further, Protégé can be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications.

3.1.2 Graphviz

Graphviz is open source graph visualization software. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. It has important applications in networking, bioinformatics, software engineering, database and web design, machine learning, and in visual interfaces for other technical domains.

The Graphviz layout programs take descriptions of graphs in a simple text language, and make diagrams in useful formats, such as images for web pages, PDF or Postscript for inclusion in other documents; or display in an interactive graph browser. Graphviz has many useful features for concrete diagrams, such as options for colors, fonts, tabular node layouts, line styles, hyperlinks, rolland custom shapes.

3.1.3 Java

Java is a programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsys-

tems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine regardless of computer architecture. Java is a general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere". Java is currently one of the most popular programming languages in use, and is widely used from application software to web applications

3.1.4 Eclipse

Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. It is written mostly in Java and can be used to develop applications in Java and, by means of various plug-ins, other programming languages including Ada, C, C++, COBOL, Perl, PHP, Python, Ruby (including Ruby on Rails framework), Scala, Clojure, and Scheme. The IDE is often called Eclipse ADT for Ada, Eclipse CDT for C/C++, Eclipse JDT for Java, and Eclipse PDT for PHP.

The initial codebase originated from VisualAge. In its default form it is meant for Java developers, consisting of the Java Development Tools (JDT). Users can extend its abilities by installing plug-ins written for the Eclipse software framework, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.

Released under the terms of the Eclipse Public License, Eclipse is free and open source software.

3.2 Using Protégé to Describe Ontology Model

An ontology describes the concepts and relationships that are important in a particular domain, providing a vocabulary for that domain as well as a

computerised specification of the meaning of terms used in the vocabulary. Ontologies range from taxonomies and classifications, database schemas, to fully axiomatised theories. In recent years, ontologies have been adopted in many business and scientific communities as a way to share, reuse and process domain knowledge. Ontologies are recently central to many applications such as scientific knowledge portals, information management and integration systems, electronic commerce, and semantic web services.

The Protégé platform supports two main ways of modelling ontologies [41]:

- The Protégé-Frames editor enables users to build and populate ontologies that are frame-based, in accordance with the Open Knowledge Base Connectivity protocol (OKBC). In this model, an ontology consists of a set of classes organised in a subsumption hierarchy to represent a domain's salient concepts, a set of slots associated to classes to describe their properties and relationships, and a set of instances of those classes - individual exemplars of the concepts that hold specific values for their properties.

- The Protégé-OWL editor enables users to build ontologies for the Semantic Web, in particular in the W3C's OWL. An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, *i.e.* facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms.

Based on the characteristics of Protégé, it is chosen to be the proper software on establishing the ontology transformer diagnosis model in this thesis. Protégé can generate and automatically classify the type of OWL. The descriptive ability of three sub-languages of OWL are enhanced in its order. When the complexity of the establishing ontology requires a more advanced sub-language description, Protégé will automatically generate the relative format

of sub-language which is required. In this thesis the relative format generated by Protégé is OWL-DL.

3.2.1 Building an OWL ontology based FRA by Protégé

The transformer diagnosis ontology is established based on FRA with the above mentioned seven-steps method. The first step is to recognize the ontology to be built is the one of transformer diagnosis based on FRA. In other words, is to ensure the area and the scope of ontology. By searching, no example of transformer diagnosis ontology based on FRA are found based on the above-mentioned research of FRA, so there is no possibility of using the existing ontology and a new ontology should be built from the beginning. The key point of building transformer fault diagnosis ontology Based on FRA is to recognize the important terms in the ontology, including classes, individuals and properties *etc.* Then define each part of ontology with top-down methods. Following these steps, an complete ontology could be built.

Because the implementation of the developed system mainly uses OWL as its ontology description language, the OWL grammar should be described in the first place while designing the database structure for ontology storage. The OWL language components include: classes, properties, data types and some relationships. As above mentioned, the key point of building ontology is to learn the structure of the ontology, establish it from the top level to the bottom level and define each level with explicit description so as to make the ontology comprehensive and all-inclusive.

In this section, FRA will be realised by Protégé. Firstly, the analysis of FRA structure is required. According to the above-mentioned introduction of FRA (Section 1.2.4 and Table 1.4), an ontology structure chart of transformer diagnosis by FRA as shown in Figure 3.1 can be obtained.

- **Creating a new OWL project**
  Firstly, start Protégé, then a **New Project** dialog box will be appeared, select **OWL Files** from the **Project Format** list section on the left

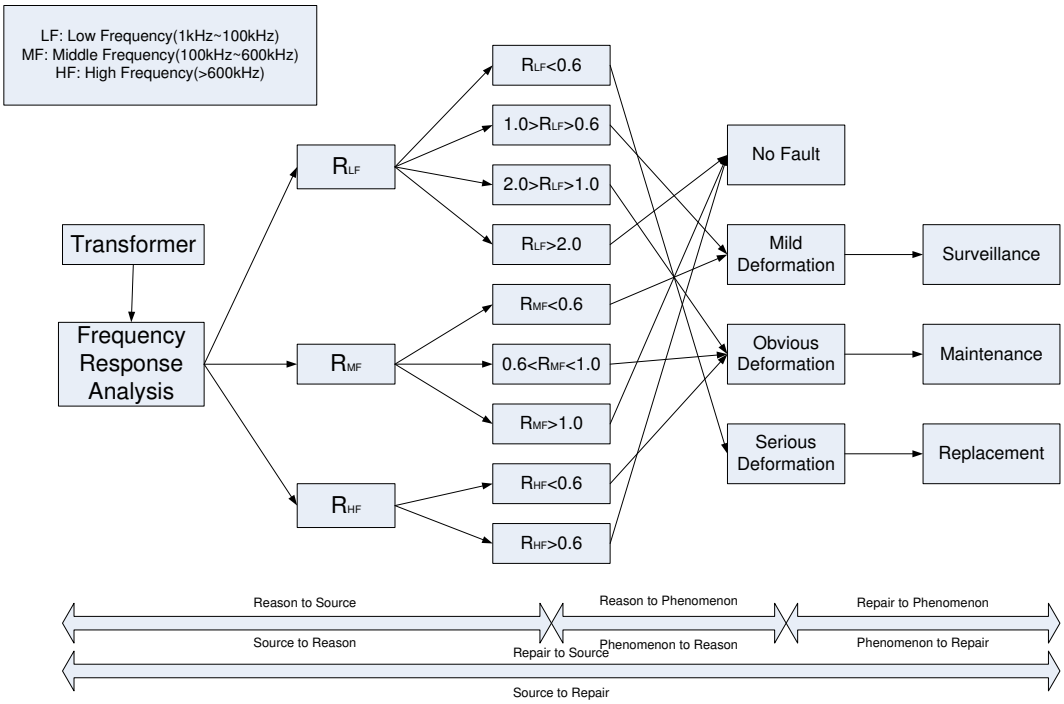

Figure 3.1: The structure of FRA.

Table 3.1: Classes in Protégé

	In Protégé
Fault Phenomenon	Fault Phenomenon
Fault Reason	Fault Reason
Fault Source	Fault Source
Repair Suggestion	Repair Suggestion
R_{LF}	Value of Winding Resistance in Low Frequency
R_{MF}	Value of Winding Resistance in Medium Frequency
R_{HF}	Value of Winding Resistance in High Frequency

hand side of the dialog box, and press **New**. After a short amount of time, a new empty Protégé.-OWL project have been created. When Protégé starts the **OWL Classes** tab shown in Figure 3.2 will be visible. The initial class hierarchy tree view should resemble the picture shown in

Figure 3.3. The empty ontology contains one class called **owl:Thing**. As mentioned previously, OWL classes are interpreted as sets of individuals (or sets of objects). The class **owl:Thing** is the class that represents the set containing all individuals. Because of this all classes are subclasses of **owl:Thing**.

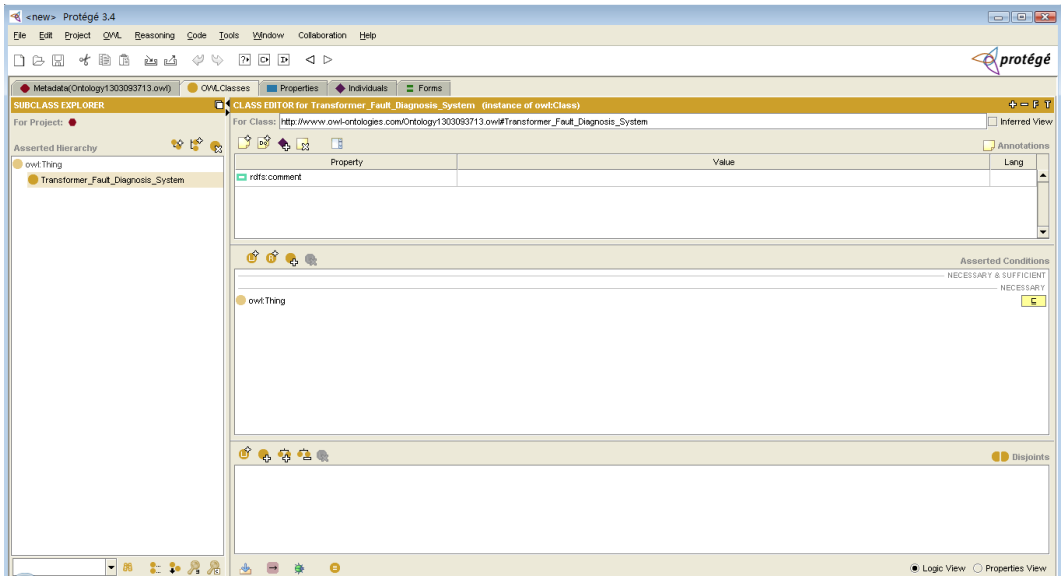

Figure 3.2: The class tab.

- **Creating a class**

  Press the **Create subclass** button shown in Figure 3.3. This button creates a new class named **Transformer Fault Diagnosis System** as a subclass of the selected class (in this case, a subclass of **owl:Thing** should be created).

- Creating some subclasses of **Transformer Fault Diagnosis System**

  Repeat the previous steps to add the classes **Fault Phenomenon**, **Fault Reason**, **Fault Source** and also **Repair Suggestion**, ensuring that **Transformer Fault Diagnosis System** is selected before the **Create subclass** button is pressed, so that the classes are created as subclasses of **Transformer Fault Diagnosis System** shown in Figure 3.4.

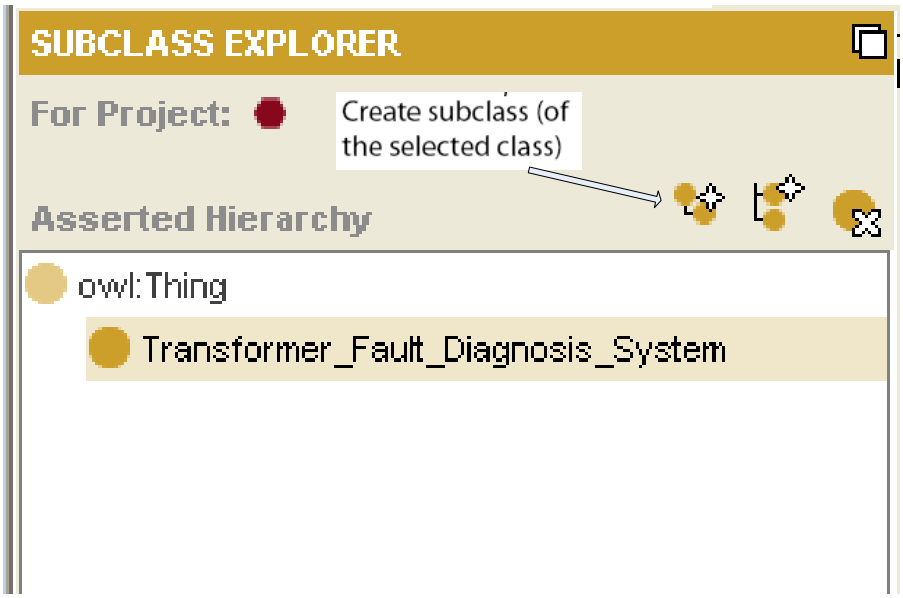

Figure 3.3: The class hierarchy pane.

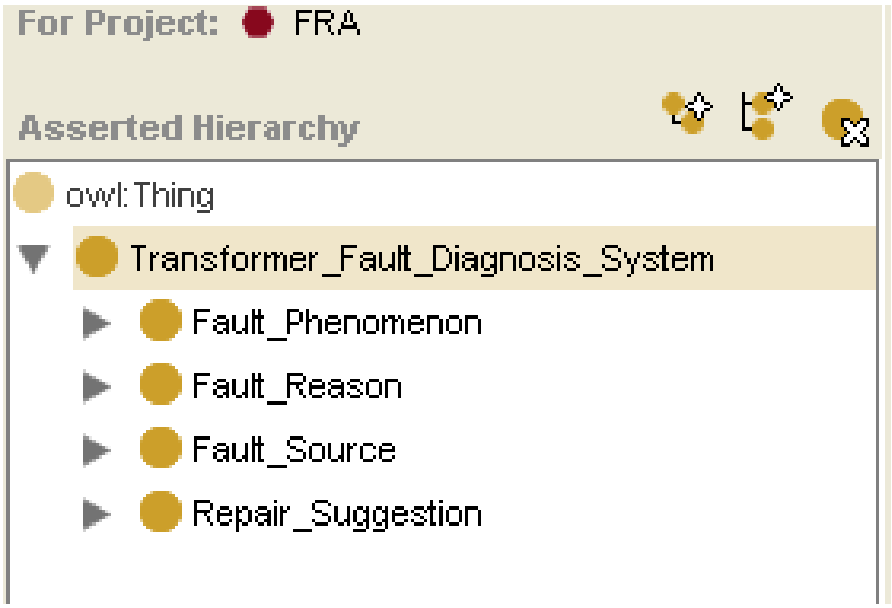

Figure 3.4: The initial class hierarchy.

- **Creating some subclasses**

  Select the class **Fault Reason** in the class hierarchy. In the same man-

ner as before, add the following subclasses of **Fault Reason**: **Value of Winding Resistance in High Frequency**, **Value of Winding Resistance in Low Frequency** and **Value of Winding Resistance in Medium Frequency**. The class hierarchy should now look similar to that shown in Figure 3.5.

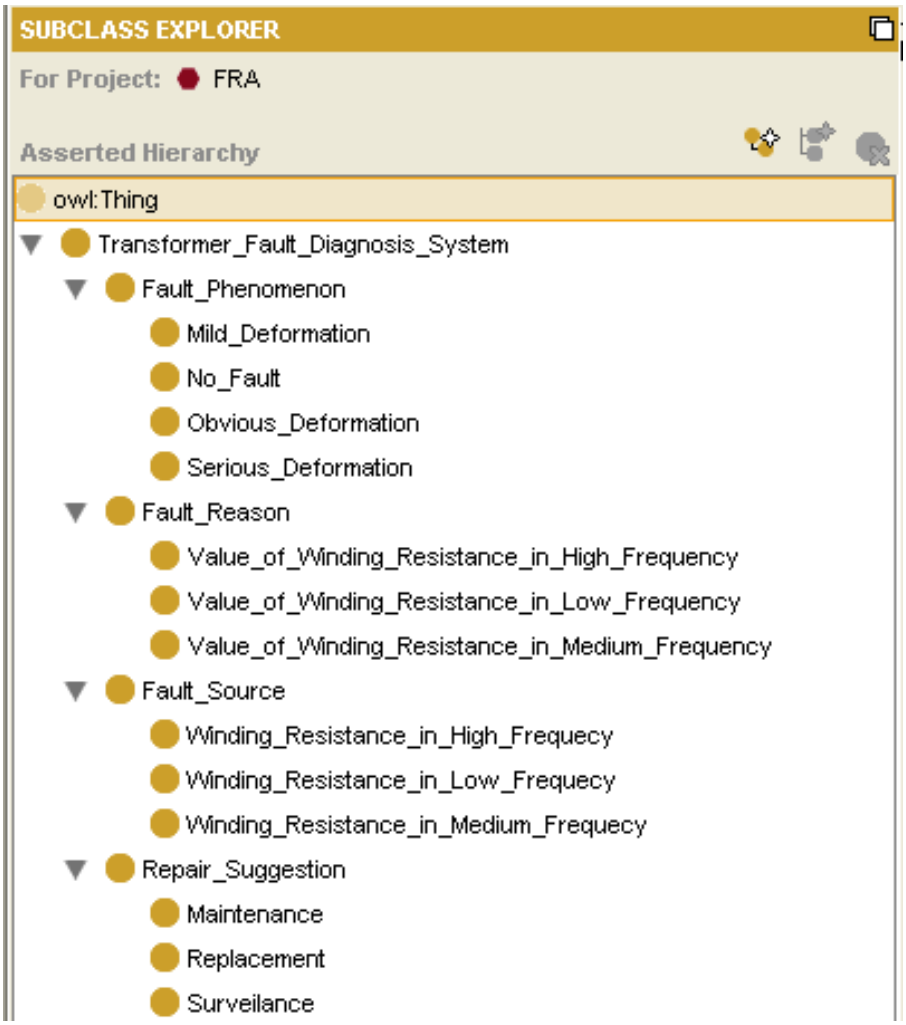

Figure 3.5: The class hierarchy.

- **Creating an object property**

  Switch to the **Propterties** tab. Use the **Create Object Property** button to create a new object property. An object property with a generic

name will be created. Rename the property to **Phenomenon to Reason**. Repeat the previous steps to add the properties **Phenomenon to Repair**, **Phenomenon to Source**, **Reason to Phenomenon**, **Reason to Repair**, **Reason to Source**, **Repair to Phenomenon**, **Repair to Reason**, **Repair to Source**, **Source to Phenomenon**, **Source to Reason** and **Source to Repair** shown in Figure 3.6.

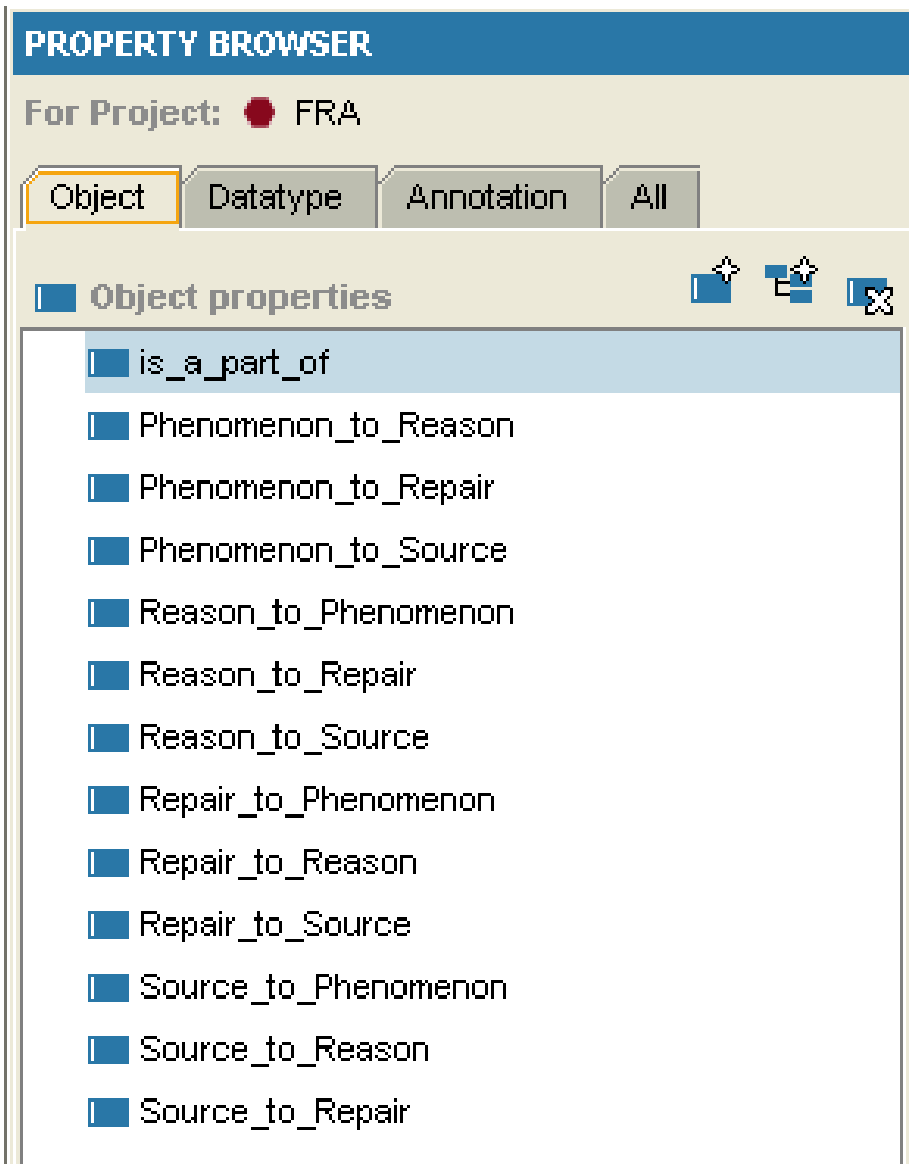

Figure 3.6: The property hierarchy.

- **Specifying the range and domain of Phenomenon to Reason**
  Make sure that the **Phenomenon to Reason** property is selected in the property hierarchy on the **Properties** tab. Press the **Add named class** button on the **Domain** widget. A dialog will appear that allows a class to be selected from the ontology class hierarchy. Select **Fault Phenomenon** and press the **OK** button. **Fault Phenomenon** should now be displayed in the domain list. Press the **Add named class** button on the **Range** widget. A dialog will appear that allows a class to be selected from the ontology class hierarchy. Select **Fault Reason** and press the **OK** button. **Fault Reason** should be displayed in the range list shown in Figure 3.7. In the same manner as before, specify the range and domain of following properties **Phenomenon to Repair**, **Phenomenon to Source**, **Reason to Phenomenon**, **Reason to Repair**, **Reason to Source**, **Repair to Phenomenon**, **Repair to Reason**, **Repair to Source**, **Source to Phenomenon**, **Source to Reason** and **Source to Repair**.

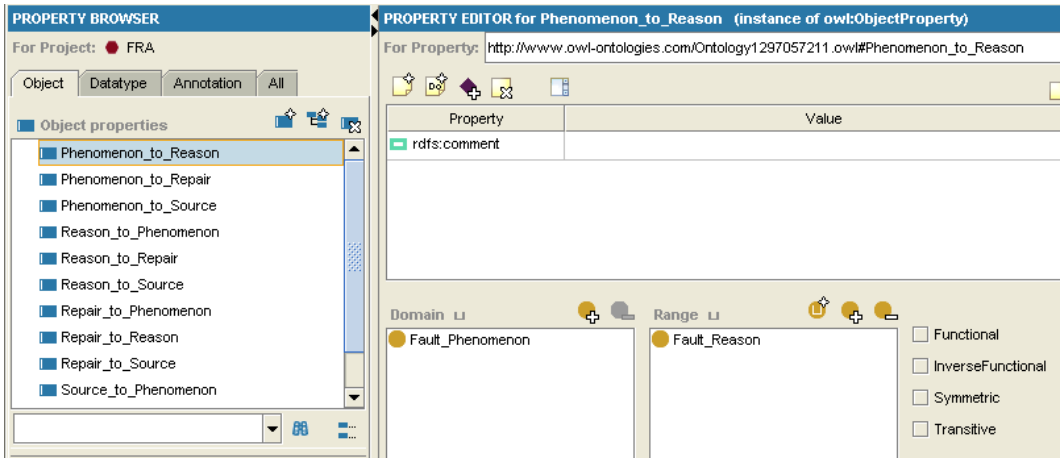

Figure 3.7: The property's domain and range.

- **Creating an individual**
  Create **Fault Phenomenon** as a subclass of **Transformer Fault Diagnosis System**. Switch to the **Individuals** Tab and select the class

Fault Phenomenon in the **Classes** tree. The press the **Create Instance** button, an individual that is a member of **Fault Phenomenon** will be created with a auto-generated name. Rename the individual using the **Name widget** to **mild deformation**. Use the above steps to create some more individuals that are members of the class **Fault Phenomenon** called **obvious deformation**, **serious deformation** and **no fault** shown as Figure 3.8. In the same manner as before, add the the following instances of **Value of Winding Resistance in High Frequency**, **Value of Winding Resistance in Low Frequency**, **Value of Winding Resistance in Medium Frequency**, **Fault Source** and **Repair Suggestion**: **Rhf is bigger than 0.6**, **Rhf is less than 0.6**, **Rlf is bigger than 1.0**, **Rlf is less than 1.0**, **Rlf is bigger than 0.6**, **Rlf is less than 0.6**, **Rlf is bigger than 2.0**, **Rlf is less than 2.0**, **Rmf is bigger than 1.0**, **Rmf is less than 1.0**, **Rmf is bigger than 0.6**, **Rmf is less than 0.6**, **winding resistance in HF**,**winding resistance in MF**, **winding resistance in LF**, **need to be maintained**, **need to be replaced** and **need to be surveilled**.

- **Defining the relation of mild deformation**
  Make sure that the instance **mild deformation** is selected in the property hierarchy on the **Individuals** tab. Press the **Add resources** button on the **Phenomenon to Reason** widget. A dialog will appear that allows a instance to be selected from the ontology instance hierarchy. Select **Rlf is bigger than 0.6**, **Rlf is less than 1.0** and **Rmf is less than 0.6**, then press the **OK** button. Press the **Add resources** button on the **Phenomenon to Repair** widget. Select **need to be surveilled** and press the **OK** button. Press the **Add resources** button on the **Phenomenon to Source** widget. Select **winding resistance in LF** and **winding resistance in MF**, then press the **OK** button. All relations based on **mild deformation** should be displayed Figure 3.9. Repeat the previous steps to define all relations of each instance based on Figure 3.1.

By measuring the value of winding under low frequency, medium frequency

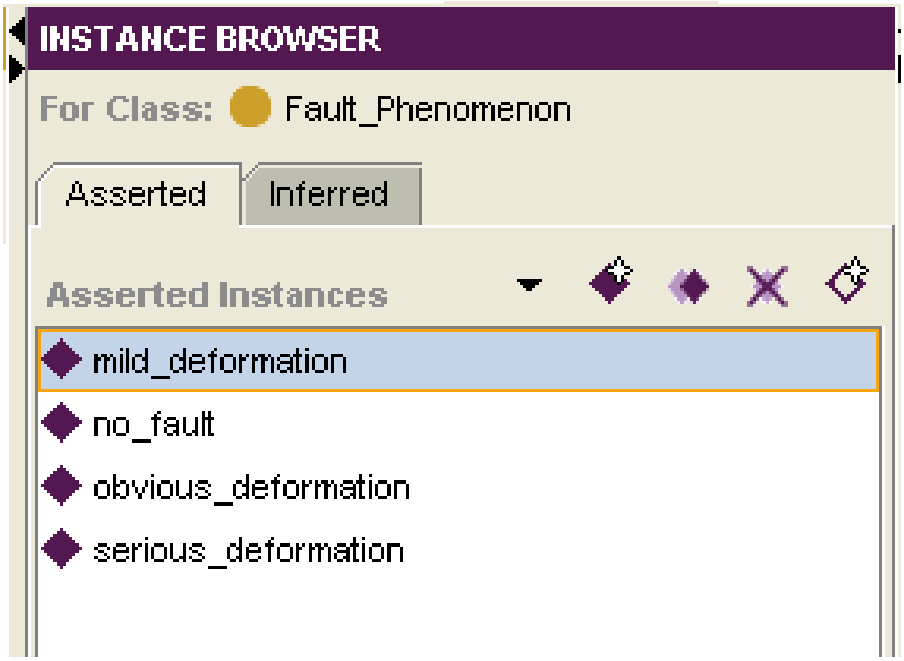

Figure 3.8: The initial instance hierarchy.

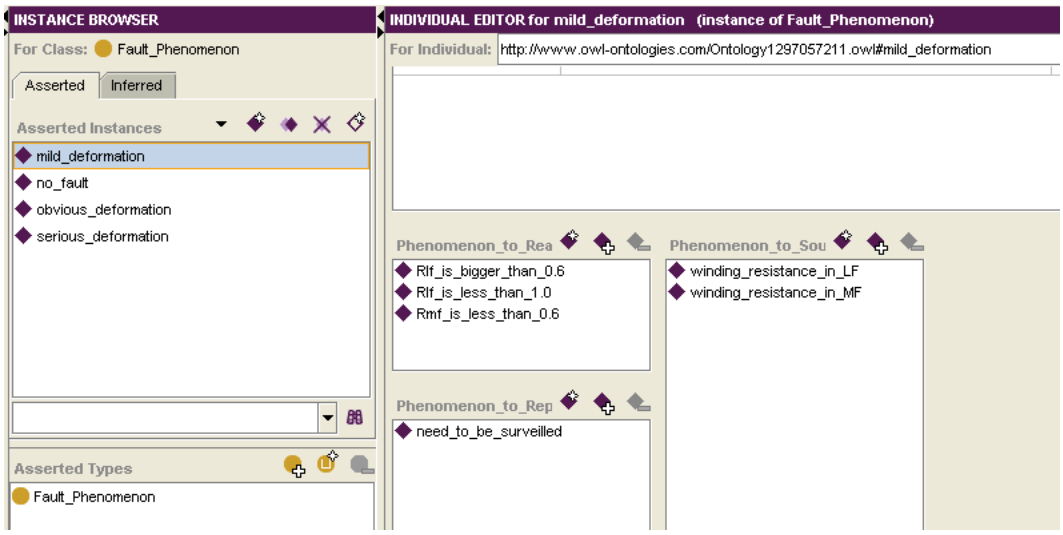

Figure 3.9: The relation of instance.

and high frequency conditions, the fault phenomena of transformer can be analysed and the repair suggestion can be provided. *e.g.* When the value

of winding is more than 1 and less than 2 under low frequency conditions, obvious deformation could appear on transformer and the repair suggestion given by this system is maintenance. The same fault phenomenon may happen under medium frequency condition and high frequency condition as well. In the Figure 3.1, every unit represents an individual. The routes between individuals are their relations. Every individual can be defined as a class. A class and all of its paratactic classes build up their upper-leveled class, which is named as father class. The lower-leveled class can be named as subclass. Every class can be defined as the subclass of its upper-leveled classes, as well as the father class of its lower-leveled classes. The classes and individuals can be defined according to the rule of Table 3.1 and Table 3.2 as shown in Figure 3.5 and Figure 3.10 in Protégé.

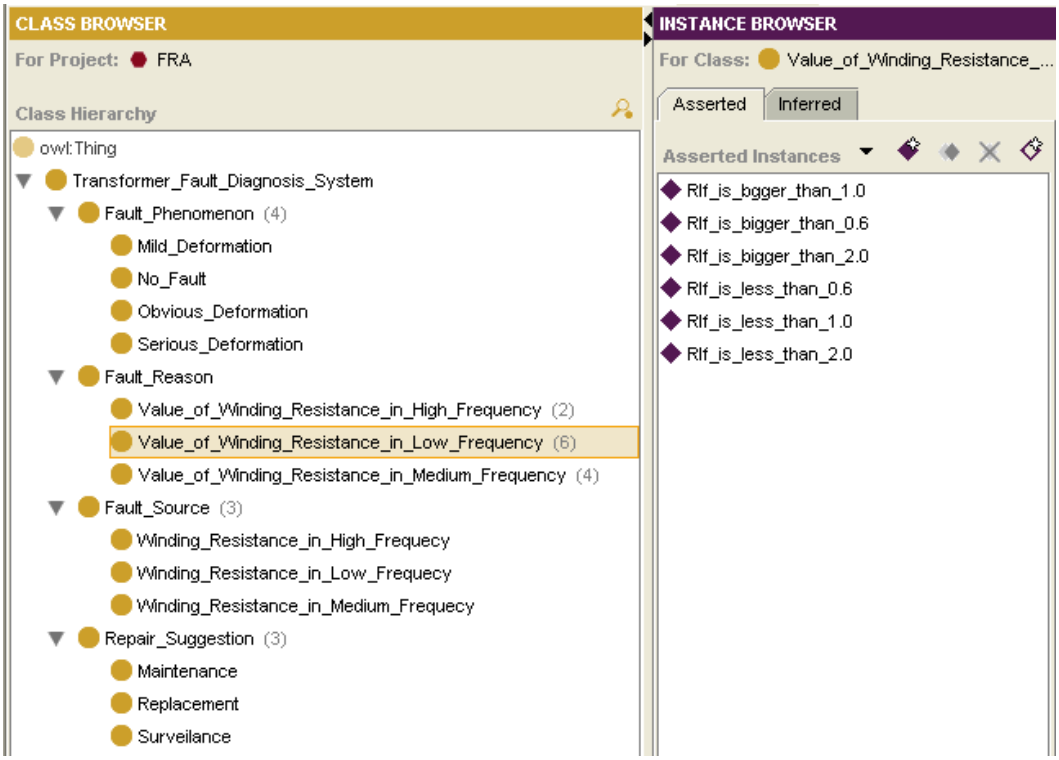

Figure 3.10: Screen-shot of the definitions of individuals from Protégé.

Figure 3.5 lists the classes and subclasses in the FRA ontology model.

Table 3.2: Individuals in Protégé

	In Protégé
R_{LF}	winding resistance in LF
R_{MF}	winding resistance in MF
R_{HF}	winding resistance in HF
R_{LF}<0.6	Rlf is less than 0.6
R_{LF}>0.6	Rlf is bigger than 0.6
R_{LF}<1.0	Rlf is less than 1.0
R_{LF}>1.0	Rlf is bigger than 1.0
R_{LF}<2.0	Rlf is less than 2.0
R_{LF}>2.0	Rlf is bigger than 2.0
R_{MF}<0.6	Rmf is less than 0.6
R_{MF}>0.6	Rmf is bigger than 0.6
R_{MF}<1.0	Rmf is less than 1.0
R_{MF}>1.0	Rmf is bigger than 1.0
R_{HF}<0.6	Rhf is less than 0.6
R_{HF}>0.6	Rhf is bigger than 0.6
No Fault	no fault
Mild Deformation	mild deformation
Obvious Deformation	obvious deformation
Serious Deformation	serious deformation
Surveillance	need to be surveilled
Maintenance	need to be maintained
Replacement	need to be replaced

The class **Transformer Fault Diagnosis System** includes four subclasses. There are subclass **Fault Phenomenon**, subclass **Fault Reason**, subclass **Fault Source** and subclass **Repair Suggestion**. These subclasses can be defined as the subclasses of its upper-leveled classes, as well as the father class of its lower-leveled classes. *e.g.* **Value of Winding Resistance in High Frequency**, **Value of Winding Resistance in Medium Frequency** and

Value of Winding Resistance in Low Frequency are subclasses of the class **Fault Reason**. Combining with Figure 3.10, **Rlf is bigger than 0.6**, **Rlf is bigger than 1.0**, **Rlf is bigger than 2.0**, **Rlf is less than 0.6**, **Rlf is less than 1.0** and **Rlf is less than 2.0** are individuals of the class **Value of Winding Resistance in Low Frequency**.

The basic structure of a fault diagnosis system is generally composed of four parts: fault phenomenon, fault reason, repair suggestion and fault source. Each of them can be defined as a class in Protégé. The relations of these four parts have already been defined in Section 2.2. In order to ensure that the users operate the stablished diagnosis system more conveniently, the relations of these four parts in Protégé can be defined as shown in Table 3.3. The new relations of these four parts can be defined as shown in Figure 3.6.

Table 3.3: Relations in Protégé description

	In Protégé
as result of	Reason to Phenomenon
result in	Phenomenon to Reason
reason to repair	Reason to Repair
repair to reason	Repair to Reason
repairing	Suggestion to Source
for repair	Source to Suggestion
have phenomenon	Source to Phenomenon
phenomenon on	Phenomenon to Source
to solve	Suggestion to Phenomenon
solve via	Phenomenon to Suggestion
affairs on	Reason to Source
have affair	Source to Reason
is a part of	is a part of

Figure 3.6 lists thirteen relations which have been redefined. Among these, twelve of the relations are reversible and corresponding in pairs, so their reversible relations can be defined with Protégé. **is a part of** is a transitive

relation by which can relate different classes and individuals together.

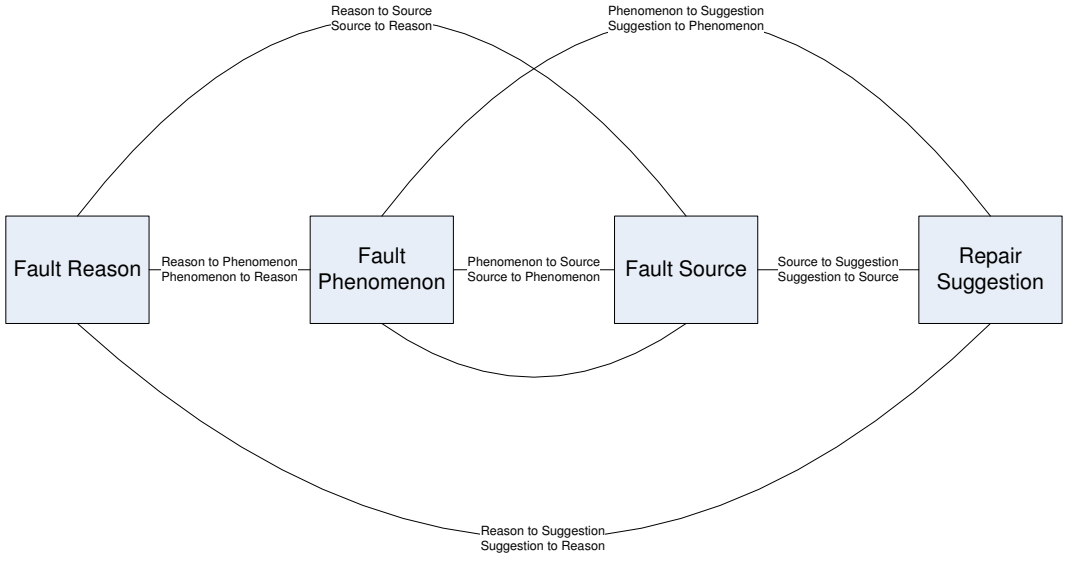

Figure 3.11: The structure of classes and relations in Protégé.

Figure 3.11 shows the relations of these four classes in reality, because the relations of classes and individuals are transitive, the relations of these four classes are the same as those of these four individuals respectively in four classes. To establish the enquiry system, definition of individuals relations in four classes is necessary. Their relations can be defined by individual editor which is a plug-in software of protege. Because each individual is not merely related to a unique individual nor class, it is very necessary to comprehensively defined all the individuals so as to ensure the veracity of this ontology enquiry system. *e.g.* individual **winding resistance in LF** is related to **mild deformation**, **no fault**, **obvious deformation** and **serious deformation** with relation **Source to Phenomenon**, **winding resistance in LF**, **winding resistance in LF** and **winding resistance in LF** with relation **Source to Repair**, **Rlf is bgger than 1.0**, **Rlf is bigger than 0.6**, **Rlf is bigger than2.0**, **Rlf is less than 0.6**, **Rlf is less than 1.0** and **Rlf is less than 2.0** with realtion **Source to Reason** as shown in Figure 3.12.

By connecting different classes and individuals through these redefined re-

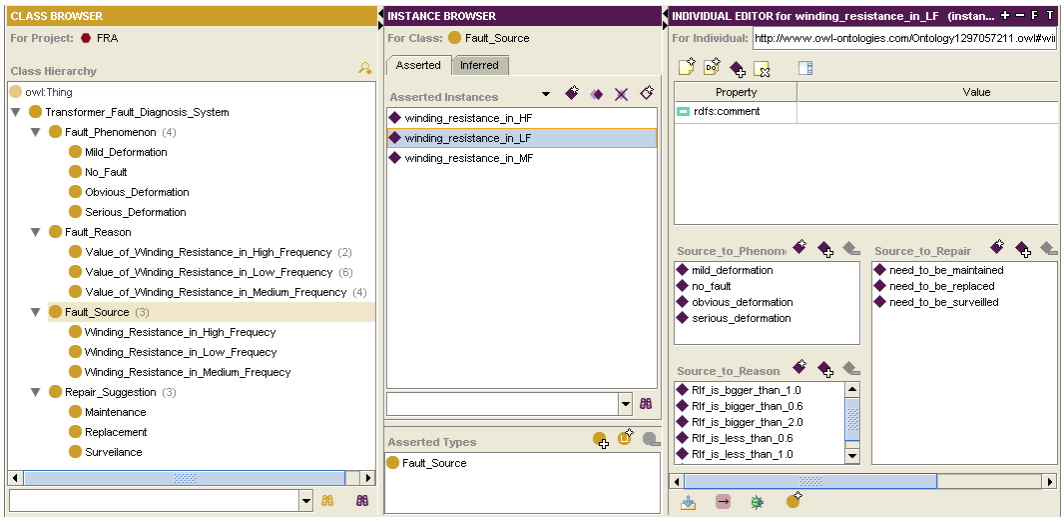

Figure 3.12: Screen-shot of the definitions of individuals and relations from Protégé.

lations, a primary transformer diagnosis model can be built based on FRA with Protégé. This model can realize the fault source locating, fault diagnosis and fault repair, *etc.*, based on different fault phenomena under FRA method.

When finished the ontology model for transformer diagnosis which based on FRA, a relation chart of the entire ontology model can be generated with Graphviz in Protégé. In Figure 3.13, the linking line represent the is a part of relation which is a transitive relation. Every ends of each linking line is an individual. Figure 3.13 may help the users understand this established ontology model with a macro scope.

3.2.2 The description of a ontology-based fault diagnosis for power transformers by Protégé

In Section 3.2.1, an ontology model for transformer diagnosis based on FRA method has been established. As above mentioned, besides FRA , there are three other methods of transformer diagnosis, TM, DGA and PDA. In this section, some more ontology models will be built up using these three methods other than FRA and these models will be integrated to be a power transformer

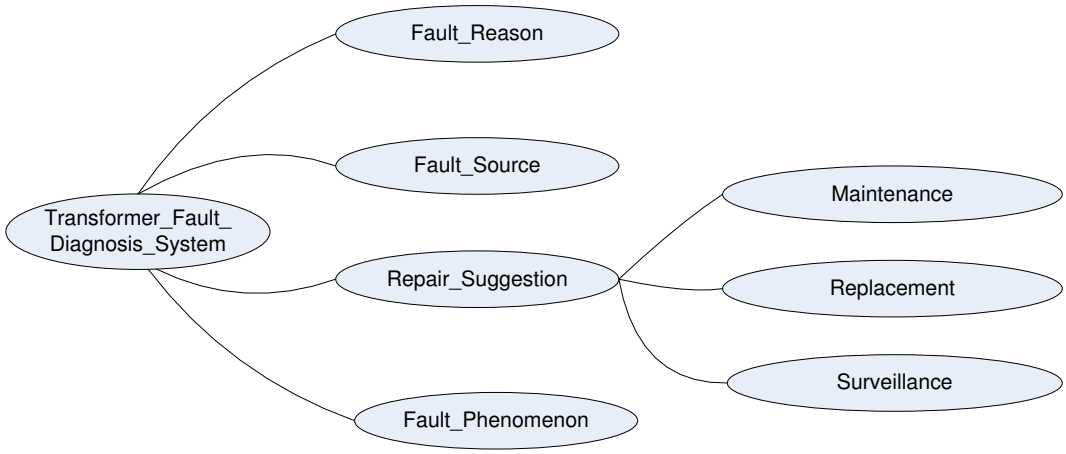

Figure 3.13: The structure of classes and individuals in the fault diagnosis ontology model.

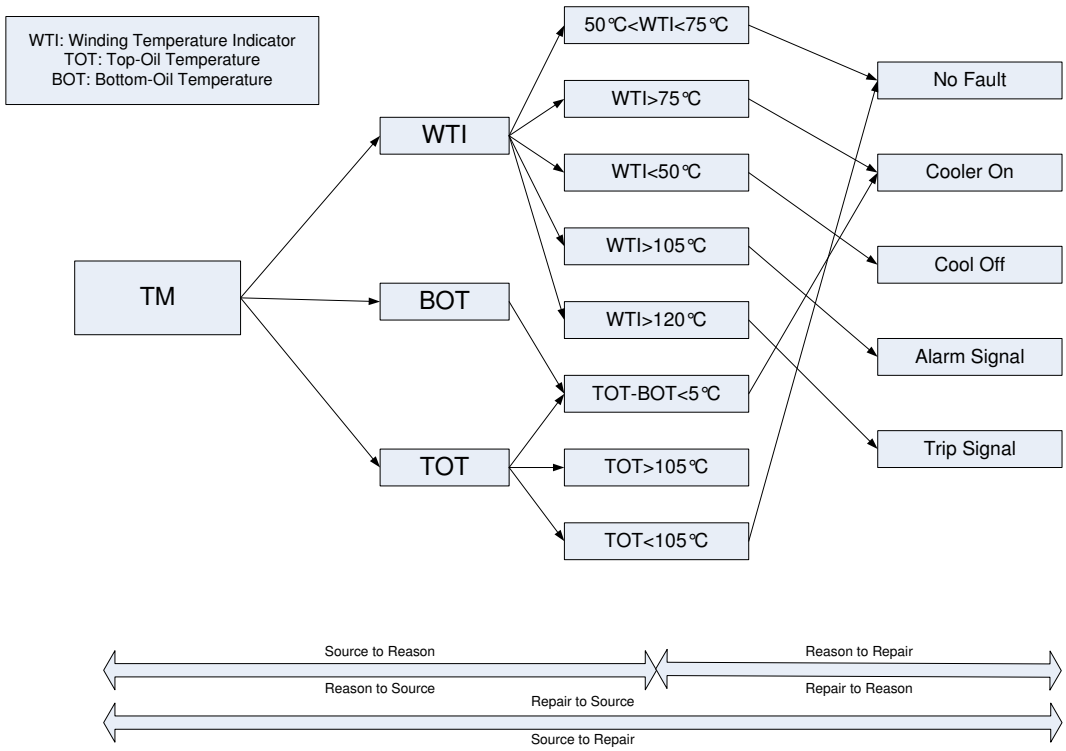

Figure 3.14: The structure of TM.

diagnosis system.

Through the introduction of Section 1.2.1 and Section 1.2.2 in Chapter 1

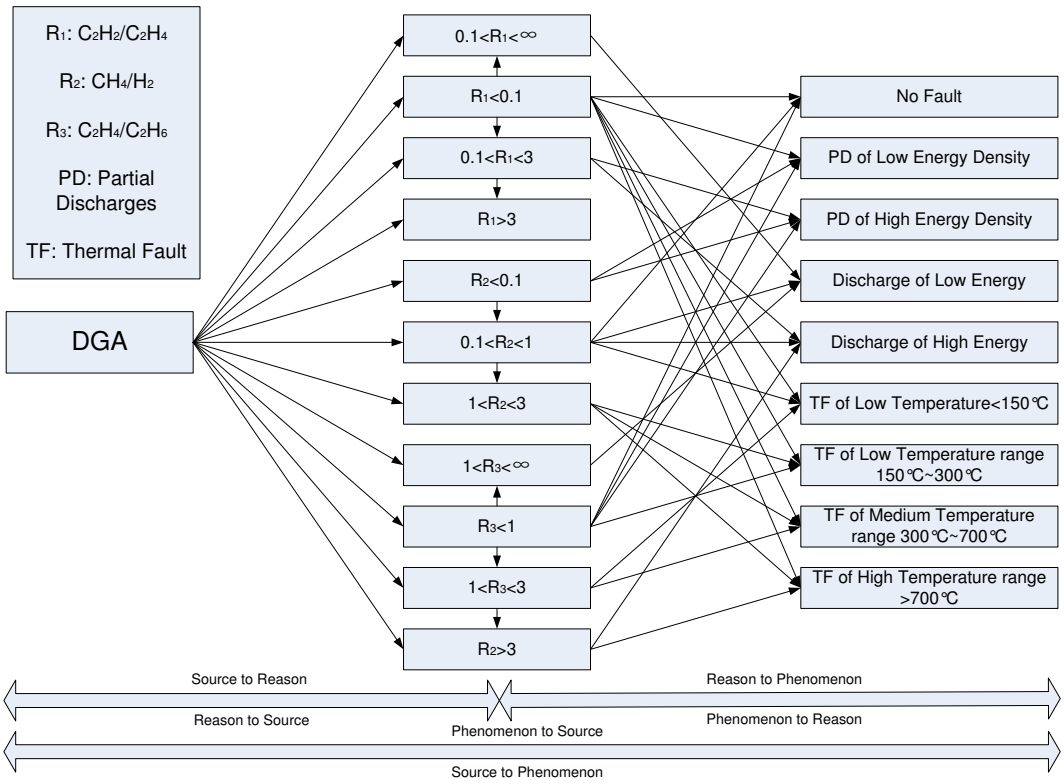

Figure 3.15: The structure of DGA.

as well as the method in last section, the structure of TM and method DGA can be derived as shown as Figure 3.14 and Figure 3.15. The logic descriptions of these transformer diagnosis methods are accordant with each other and all these methods contain one or more elements of transformer diagnosis (fault source, fault phenomenon, fault reason and repair suggestion), therefore these methods can be integrated in OWL with Protégé so as to build up an integrated transformer diagnosis system.

Figure 3.17 demonstrates the whole structure of transformer diagnosis system, which includes the established transformer diagnosis ontology based on FRA. According to this structure, the FRA ontology model can be supplemented and consummated. Based on the rule from last section, this ontology system can be integrated with TM, DGA and PDA methods with Protégé so as to make it a power transformer diagnosis system contains four methods.

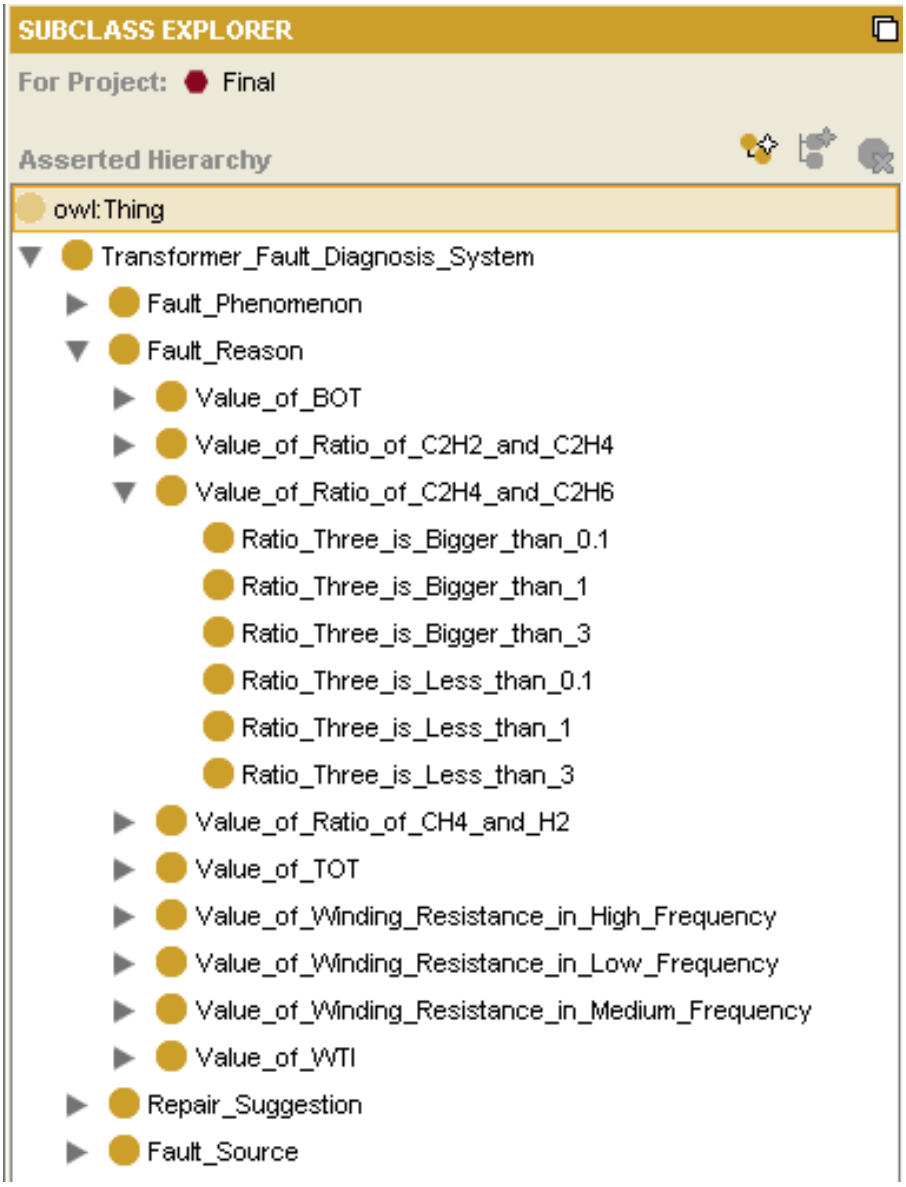

Figure 3.16: The structure of an ontology model for power transformer fault diagnosis.

This ontology-based transformer diagnosis system contains four most commonly used transformer fault diagnosis methods, it records the realistic logic description of these methods into ontology with OWL. This model contains fault phenomenon, fault reason, fault source and repair suggestion, which are all elaborated as detailed as possible in OWL. It consists of abundant detailed

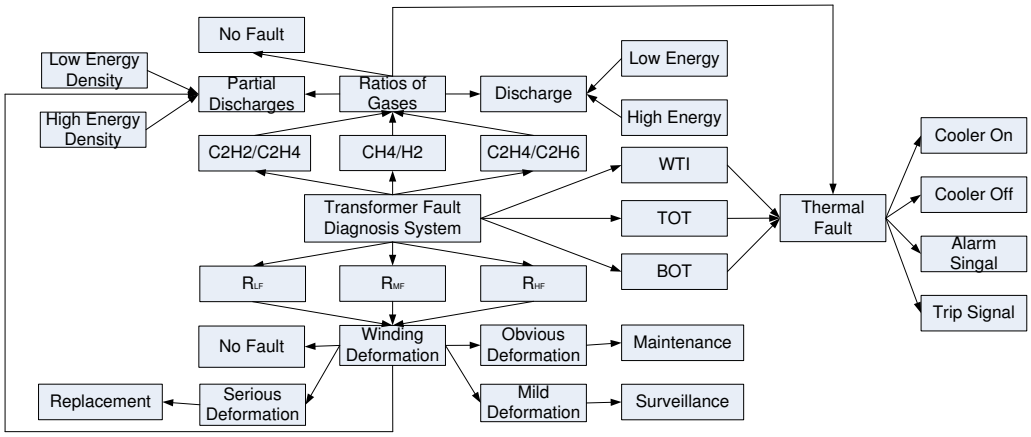

Figure 3.17: The structure of an ontology model for power transformer fault diagnosis.

data, which defines the relations of fault phenomenon, fault reason, fault source and repair suggestion clearly and provides necessary conditions for the construction of transformer diagnosis query system. Not all transformer diagnosis method provide fault phenomenon, fault reason, fault source and repair suggestion, but this does not affect the availability of building the transformer diagnosis model with Protégé. With two of four the above-mentioned elements, an expression of relation can be built up, by which the ontology model is established.

3.2.3 Query system for power transformer fault diagnosis

As is mentioned in previous sections, the existing transformer fault diagnosis and query methods run independently. By using ontology, each of these methods can be integrated into the system to improve diagnosis efficiency and accuracy. The established power transformer fault diagnosis model, which is based on ontology, can be queried with the query plug-in of Protégé. All the figures in this section are screen-shot from Protégé.

Firstly, the query purpose need to be selected. Figure 3.18 shows that there

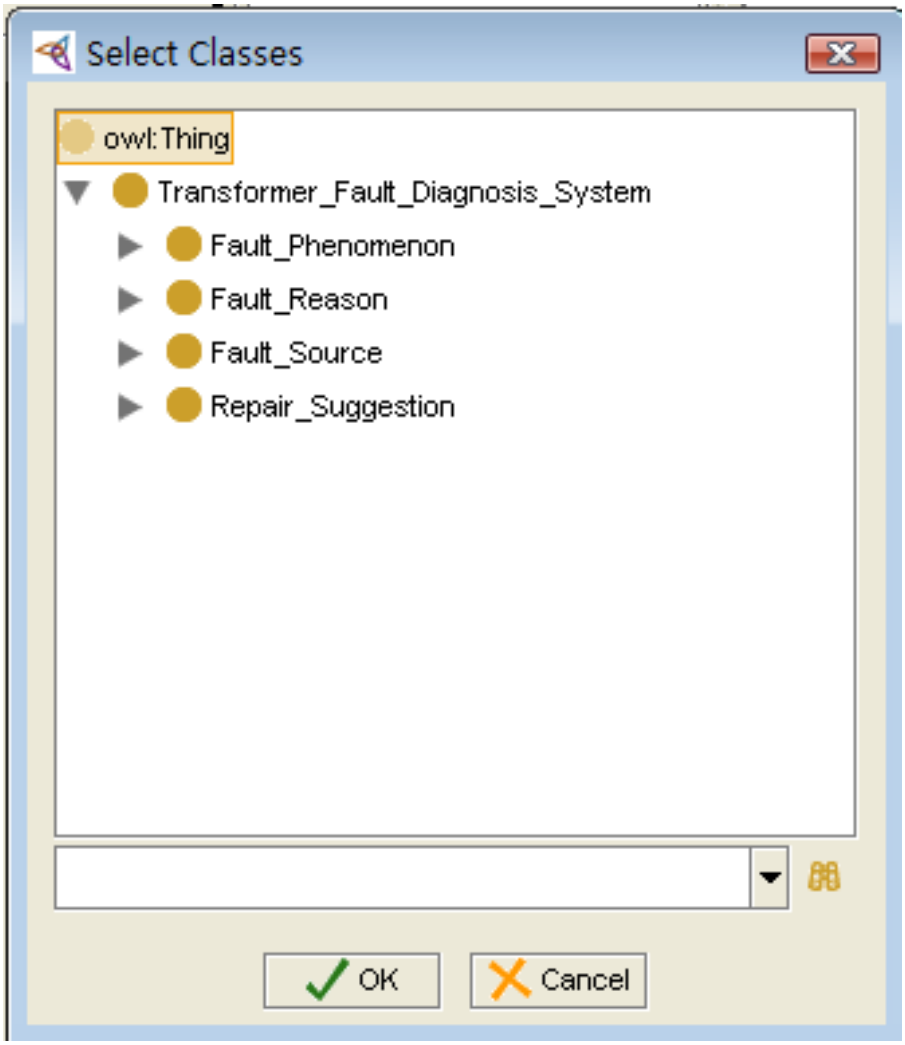

Figure 3.18: Set a query task

are five options, which can be selected in this query system (fault phenomenon, fault reason, fault source and repair suggestion). *e.g.* **Fault_Reason** should be selected if the fault diagnosis task is to identify the reason of a transformer fault.

Secondly, loading the data is related to fault reasons in the system. All the causes of faults are entered into the system as previous work. All fault reasons are shown in Figure 3.19. However these are all the causes of the malfunction, not the final outcome of the query.

Then, the relationship between a query diagnosis task and input data need

Search Results (42)

- ratio_one_is_less_than_1 (Value_of_Ratio_of_C2H2_and_C2H4)
- ratio_one_is_bigger_than_1 (Value_of_Ratio_of_C2H2_and_C2H4)
- ratio_one_is_less_than_3 (Value_of_Ratio_of_C2H2_and_C2H4)
- ratio_one_is_less_than_0.1 (Value_of_Ratio_of_C2H2_and_C2H4)
- ratio_one_is_bigger_than_0.1 (Value_of_Ratio_of_C2H2_and_C2H4)
- ratio_one_is_bigger_than_3 (Value_of_Ratio_of_C2H2_and_C2H4)
- TOT_is_less_than_105C (Value_of_TOT)
- TOT_is_bigger_than_105C (Value_of_TOT)
- TOT-BOT_is_less_than_5C (Value_of_BOT)
- TOT-BOT_is_bigger_than_5C (Value_of_BOT)
- Rlf_is_bigger_than_0.6 (Value_of_Winding_Resistance_in_Low_Frequency)
- Rlf_is_less_than_1.0 (Value_of_Winding_Resistance_in_Low_Frequency)
- Rlf_is_less_than_2.0 (Value_of_Winding_Resistance_in_Low_Frequency)
- Rlf_is_bgger_than_1.0 (Value_of_Winding_Resistance_in_Low_Frequency)
- Rlf_is_less_than_0.6 (Value_of_Winding_Resistance_in_Low_Frequency)
- Rlf_is_bigger_than_2.0 (Value_of_Winding_Resistance_in_Low_Frequency)
- ratio_three_is_less_than_3 (Value_of_Ratio_of_C2H4_and_C2H6)
- ratio_three_is_less_than_1 (Value_of_Ratio_of_C2H4_and_C2H6)
- ratio_three_is_less_than_0.1 (Value_of_Ratio_of_C2H4_and_C2H6)
- ratio_three_is_bigger_than_1 (Value_of_Ratio_of_C2H4_and_C2H6)
- ratio_three_is_bigger_than_0.1 (Value_of_Ratio_of_C2H4_and_C2H6)
- ratio_three_is_bigger_than_3 (Value_of_Ratio_of_C2H4_and_C2H6)

Figure 3.19: Load the cause of the transformer fault from the database

to be linked. There are several relationships which are already entered into the system. In this case, only the fault occurred on the winding in medium frequency has been known. Therefore, select the **Reason_to_Source** in Figure 3.20 and **winding_resistance_in_MF** in Figure 3.21 and in this way, the cause of the fault and fault source are linked.

The formation of the formula is **Fault_Reason** roots in **winding_resistance_in_MF**. Finally, the cause of the transformer fault **Rmf_is_bigger_than_0.6** (winding resistance is bigger than 0.6 Ohm in the medium frequency range), **Rmf_is_less_than_0.6**, **Rmf_is_less_than_1.0** and **Rmf_is_bigger_than_1.0**

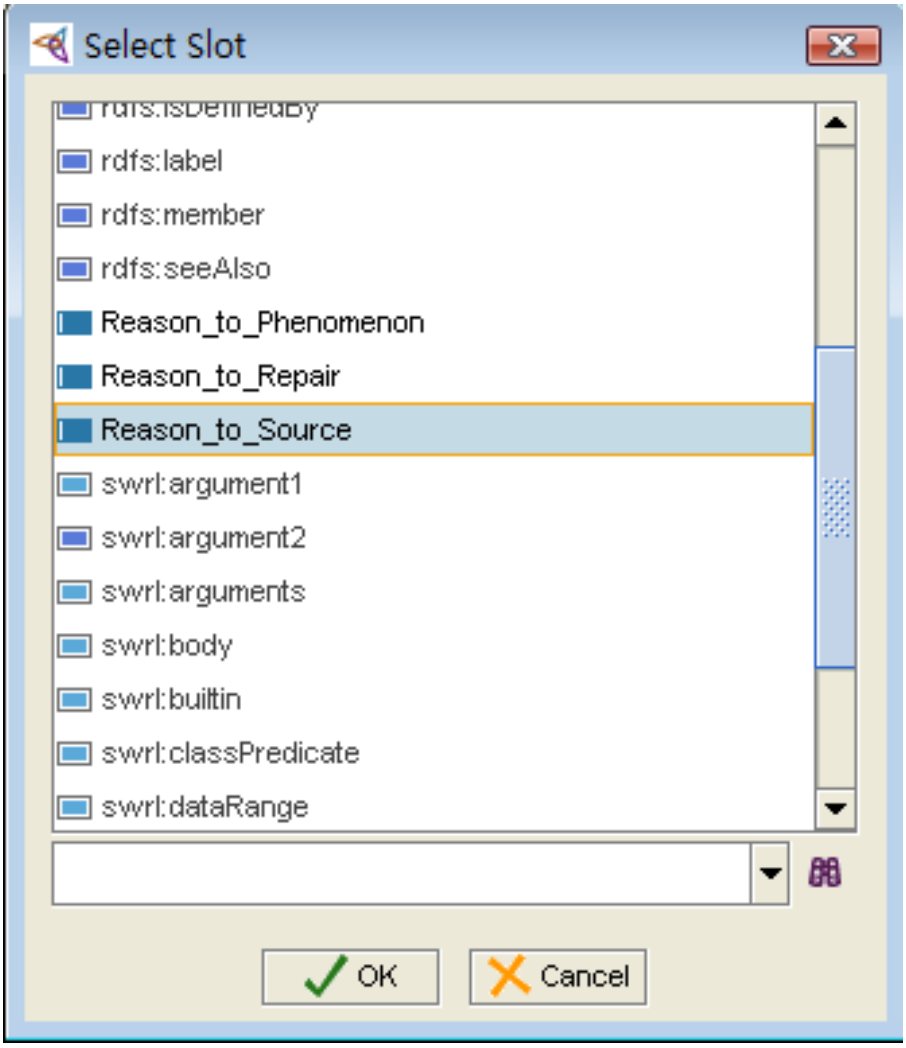

Figure 3.20: Select the relationship

is shown in Figure 3.22. However, these four causes in this example are not accurate enough.

Any query with the same diagnosis task can be assembled in order to enhance accuracy. Because fault phenomenon, fault reason, fault source, repair suggestion and fault phenomenon are all connected to each other, if more information is known in this query, it can be added to the query as the first query's supplement to improve the efficiency and veracity of the query. **Mild_Deformation** has been known in this case, which is a fault phenomenon. This condition can be added as a second query into the system. In Figure 3.23,

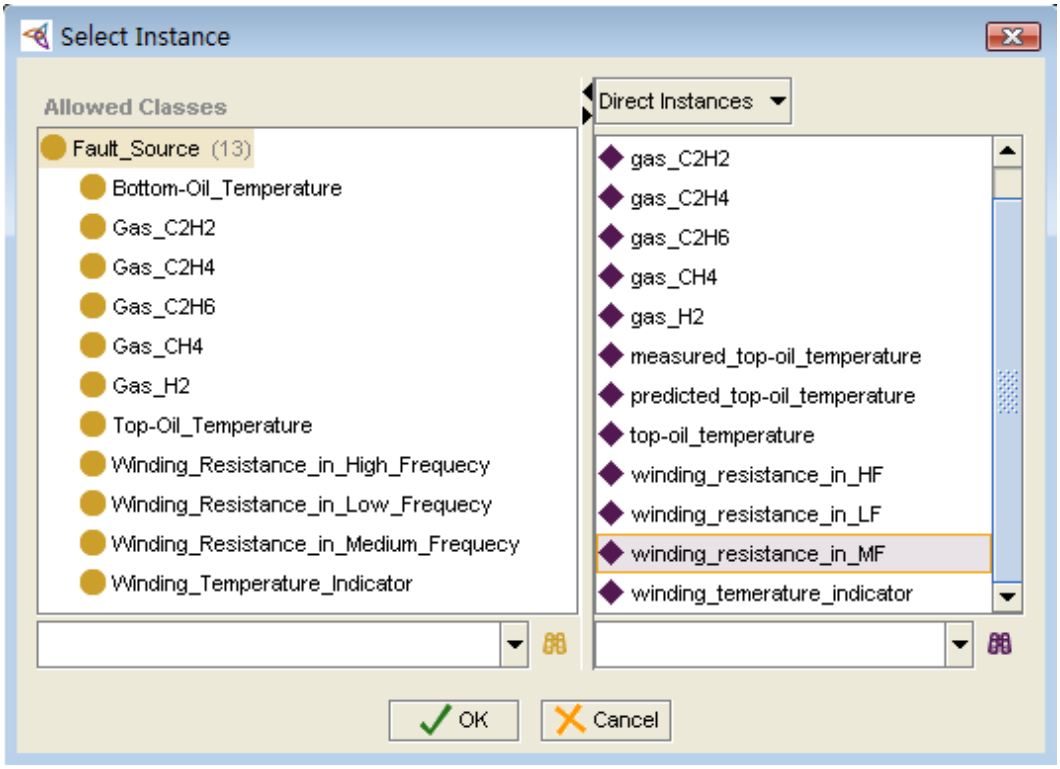

Figure 3.21: Select the fault source

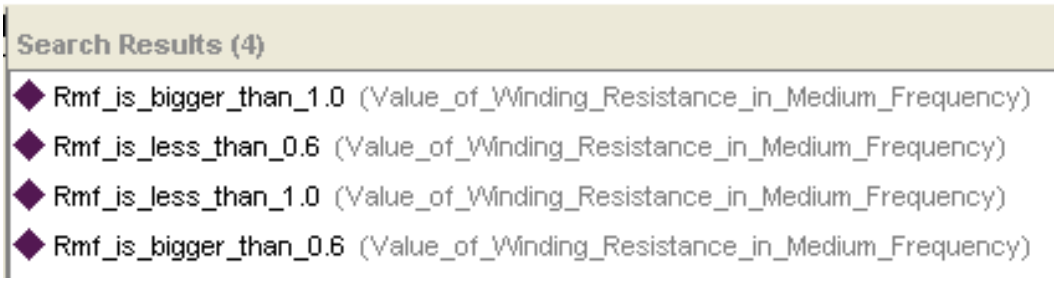

Figure 3.22: Deduce the result

select the relationship **Reason_to_Phenomenon** which links the fault phenomenon and fault reason.

An accurate result, **Rmf_is_less_than_0.6** shown in Figure 3.24 is deduced. The result is much more accurate than the previous one. Similarly, the above methods can be used to query the repair suggestion, the sources of transformer fault and the fault phenomena.

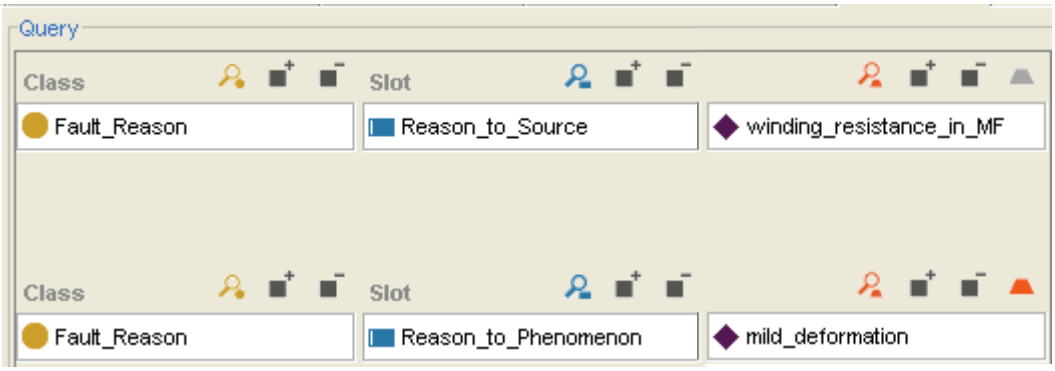

Figure 3.23: Plus a second condition of the query

Figure 3.24: Result after the second query

3.2.4 Interface

In this section, a system interface for transformer fault reasons query is established with Java and Jena. This interface allows the non-professional to query on the transformer diagnosis reasons as well as offering repair suggestions and solutions according to their fault phenomena. Jena is an open source Semantic Web framework for Java. It provides an API to extract data from and write to RDF graphs. The graphs are represented as an abstract "model". A model can be sourced with data from files, databases, URLs or a combination of these. Because Jena provides support for OWL, a query system interface can be established by using Eclipse with Java ,Jena and OWL knowledge. All the figures in this section are screen-shot from Eclipse.

Figure 3.25 lists the function package for establishing transformer faults diagnosis system interface. Jena function package is listed in the end, by adding these function packages, eclipse can quote information in OWL files built by Protégé with java. Among these, function package **public FaultRepairQuery** is used to create the interface.

```
package fault.repair;

import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.FileInputStream;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.JTextArea;
import javax.swing.UIManager;

import net.java.dev.designgridlayout.DesignGridLayout;

import com.hp.hpl.jena.ontology.Individual;
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.ontology.OntResource;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.NodeIterator;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
```

Figure 3.25: The list of function package

The statement in Figure 3.26 shows the definition of name, style and size of the query interface, at the beginning of which loaded otology with **private OntModel model** and **loadOntology**.

Figure 3.27 defines the query structure of the interface. First of all, it defines to select fault phenomenon in the drop-down box, then it queries the fault phenomenon through attribute **Phenomenon to Reason**. Finally, it queries the repair solution and suggestion with **Phenomenon to Reason**,

```
public class FaultRepairQuery {

    private static JFrame frame;
    private JComboBox FP;
    private JButton Query;
    private JTextArea FR;
    private JScrollPane FRScrollPane;
    private JTextArea RS;
    private JScrollPane RSScrollPane;
    private OntModel model;

    public FaultRepairQuery() {
        initFrame();
        initContentPane();
        initActions();
        loadOntology();
        frame.setVisible(true);
    }

    private void initFrame() {
        frame = new JFrame
        ("Ontology Model for Power Transformer Fault Diagnosis");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(600, 400);
        double width = Toolkit.getDefaultToolkit()
        .getScreenSize().getWidth();
        double height = Toolkit.getDefaultToolkit()
        .getScreenSize().getHeight();
        frame.setLocation((int) (width - frame.getWidth()) / 2,
                (int) (height - frame.getHeight()) / 2);
        frame.setResizable(false);
    }
```

Figure 3.26: Definition of name, sytle and size of interface

and show the result in the end.

In Figure 3.28, the ontology files generated by Protégé are loaded. Meanwhile it defines that if it was loaded unsuccessfully, an error message will pop up. *N.B.* the loading OWL files should be in the same folder with the files eclipse generated, otherwise the loading will fail.

Figure 3.29 shows that after loaded the OWL file which are required to query, the class of fault phenomenon should be found at first place, then all

```
private void initContentPane() {
    FP = new JComboBox();
    Query = new JButton("Query");
    Query.setEnabled(false);

    FR = new JTextArea();
    FR.setEditable(false);
    FRScrollPane = new JScrollPane(FR);

    RS = new JTextArea();
    RS.setEditable(false);
    RSScrollPane = new JScrollPane(RS);

    DesignGridLayout layout =
    new DesignGridLayout(frame.getContentPane());

    layout.row().grid
    (new JLabel("Fault Phenomenon:")).add(FP, 4)
            .add(Query);

    layout.row().center().fill().add(new JSeparator());
    layout.row().grid
    (new JLabel("Fault Reason:")).add(FRScrollPane);

    layout.row().center().fill().add(new JSeparator());
    layout.row().grid
    (new JLabel("Repair Suggestion:")).add(RSScrollPane);
}
```

Figure 3.27: Definition of structure of interface

the individuals in this class should be queried and the names all individuals will be added to the pre-defined drop-down boxes respectively.

Finally, the appearance and the entry for the procedure are shown in Figure 3.30

By running the program, the interface can be generated. Queries of the fault phenomenon can be made through this interface and the fault reasons and repair suggestions can be resulted, *e.g.* in Figure 3.31, first choosing **mild deformation** in the drop-down boxes, then press `Query` button, the query

```
    public void actionPerformed(ActionEvent e) {
        FR.setText("");
        RS.setText("");
        Individual fp = model
        .getIndividual
        ("http://www.owl-ontologies.com/unnamed.owl#"
                        + FP.getSelectedItem());
        NodeIterator p2r = fp
        .listPropertyValues(model
        .getObjectProperty
        ("http://www.owl-ontologies.com/unnamed.owl#" +
                "Phenomenon_to_Reason"));
        boolean first = true;
        while (p2r.hasNext()) {
            if (!first){FR.append("\n");}
            Resource fr = (Resource) p2r.nextNode();
            FR.append(fr.getLocalName());
            first =false;}

        NodeIterator p2rs = fp
        .listPropertyValues(model
        .getObjectProperty
        ("http://www.owl-ontologies.com/unnamed.owl#" +
                "Phenomenon_to_Repair"));
        first = true;
        while (p2rs.hasNext()) {
            if (!first){RS.append("\n");}
            Resource rs = (Resource) p2rs.nextNode();
```

Figure 3.28: Definition of error

result is shown as following: the fault reason of **mild deformation** is **Rlf is less than 1.0**, **Rmf is less than 0.6** and **Rlf is bigger than 0.6**, the repair suggestion of **mild deformation** is **need to be surveilled** are shown in Figure 3.31 and Figure 3.32

```
private void loadOntology() {
    model = ModelFactory.createOntologyModel
    (OntModelSpec.OWL_MEM);
    try {
        model.read(new FileInputStream("Final.owl"), null);
    } catch (Exception ex) {

        ex.printStackTrace();
        JOptionPane.showMessageDialog(frame, ex.getMessage(),
                "Loading Ontology Error!",
                JOptionPane.ERROR_MESSAGE);
        System.exit(0);
    }

    OntClass FPClass = model
            .getOntClass
            ("http://www.owl-ontologies.com/unnamed.owl#" +
                    "Fault_Phenomenon");

    ExtendedIterator<? extends OntResource>
    it = FPClass.listInstances();

    while (it.hasNext()) {
        OntResource fp = it.next();
        FP.addItem(fp.getLocalName());
    }
    FP.setSelectedItem(null);
}
```

Figure 3.29: Definition of fault phenomenon in interface

```
public static void main(String[] args) {
    try {
        UIManager.setLookAndFeel
        (UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        JOptionPane.showMessageDialog
        (frame, "UI Error!", "UI Error!",
        JOptionPane.ERROR_MESSAGE);
    }

    new FaultRepairQuery();
}
```

Figure 3.30: Definition of appearance and entry of procedure

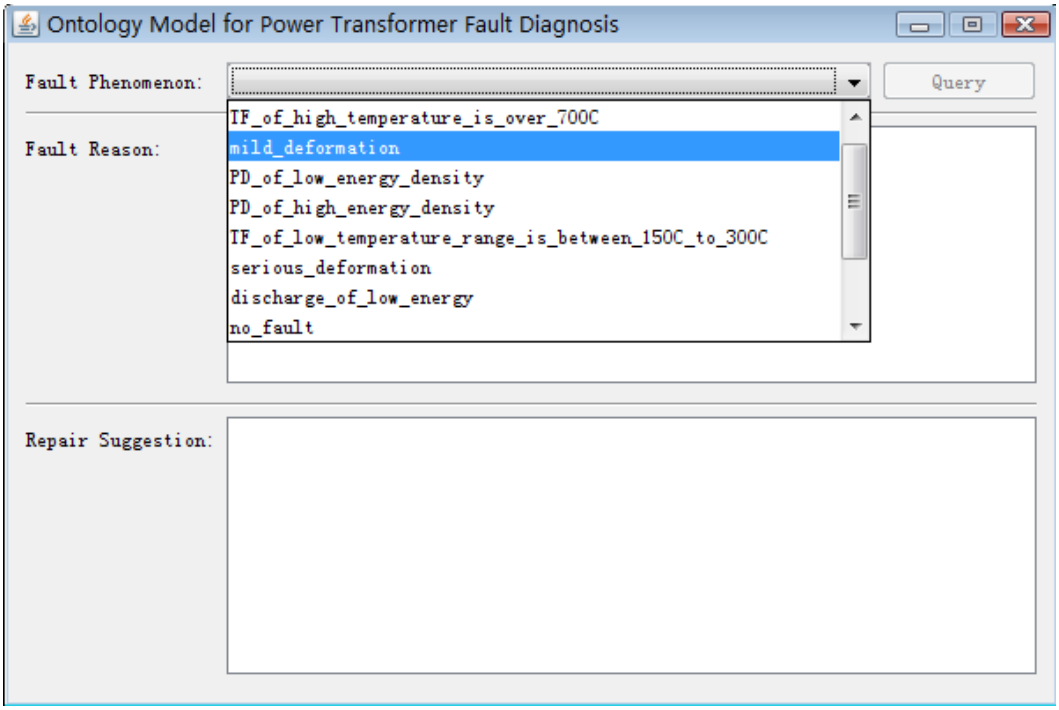

Figure 3.31: Select the fault phenomenon in interface

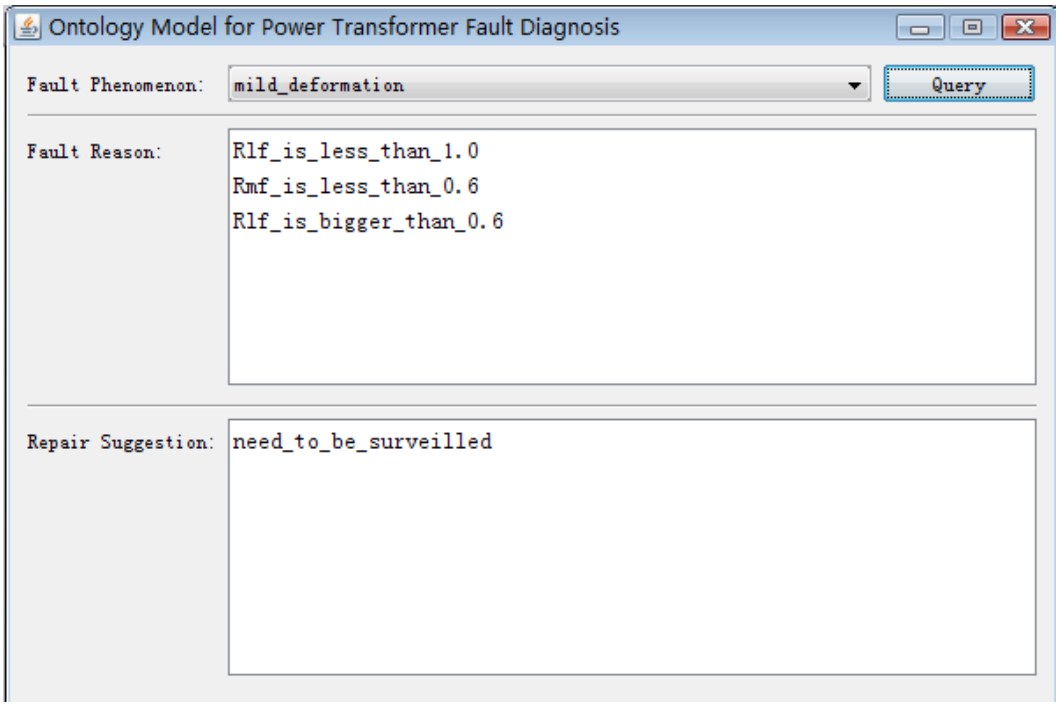

Figure 3.32: Result of query by using interface

Chapter 4

Conclusion and Future Work

This chapter concludes the thesis and summarises the major achievements of the presented research work in the field of power transformer fault diagnosis and ontology model. The following sections provide a summary of all the results obtained and the details of the major contributions. Then the advantages of applying ontology to power transformer fault diagnosis system and other diagnosis system are discussed. Finally, the suggestions for future research are presented.

4.1 Summary

This thesis has described the development of an ontology-base power transformer fault diagnosis system. In this system, TM, DGA, and FRA were integrated into ontology model. Specifically, in the preceding chapters, the following work and results were presented.

1. A background of power transformer fault diagnosis system, particularly for TM, DGA, and FRA. An introduction to ontology and OWL, including the definitions and architectures were given in Chapter 1.

2. In Chapter 2 introduced how to write ontology programs with OWL in seven-steps. Then inducted expert system and compared the difference

between traditional expert system and the ontology-based diagnosis system. The analysis of fault diagnosis system structure explained the relations between different part and elaborated the definition and application of each component of ontology. Finally, with a detailed explanation of OWL, a primary ontology model for power transformer fault diagnosis was established based on TM.

3. Firstly introduced Protégé, the software used to build ontology model. Then an ontology model was built with Protégé based on FRA. After developed and supplemented of this model, it was integrated with TM, DGA, and FRA of ontology by which it extend to be a query system aims at power transformer fault diagnosis. Finally, based on the ontology model, an interface for the ontology-based power transformer fault diagnosis system was built up by Java.

4.2 Advantages

In this thesis, the benefits of applying ontology technology in power transformer fault diagnosis has been described. There are many advantages of the application of ontology. These advantages are summarised in four areas, including expansibility, open, wide application and compatibility, which are introduced as follows.

Expansibility

The ontology fault diagnosis model consists of four methods of transformer fault diagnosis. Besides these, there are many other ways in transformer fault diagnosis. With the development of this area, more and more new methods will be introduced and applied in transformer fault diagnosis in future. The majority of the basic structure of transformer fault diagnosis method are very similar, they are all consist of fault phenomenon, fault reason, repair suggestion and fault source. If the supplement of the model in-use is required, it is just need to integrate and consummate new methods in the model according to the

same rules. And one of the advantages of this ontology model is that these new methods could be used and integrated by ontology model.

Open

Protégé is the software used to build this ontology model for fault diagnosis. So other scholars and researchers who are interested in transformer fault diagnosis could join in the research and perfection of this model. One very important advantage for consummate the system is that this model is an open system and the relevant softwares are all free.

Widely applied

The purpose for designing this model is for transformer fault diagnosis. As mentioned earlier, ontology could be used in defining almost anything in reality. Thus the model which is built by ontology technology could also be applied in the fault diagnosis in other electronic areas. *e.g.* the approach of establishing this model can also be applied in medication area, such as for the disease diagnosis and treatment. Meanwhile, this model can also be used in the book enquiry of library.

Compatibility

The existing expert system could only use four methods to diagnose transformer fault. The current models can only be used under some specific conditions, and it should satisfy pre-configured model rules for diagnosis. With the development and perfection of this system, it is possible that this model could be used under any condition in transformer fault diagnosis.

4.3 Suggestions for Future Work

There are many existing diagnosis methods for transformer, *e.g.* expert-system and data mining. Based on the analysis of advantage and disadvantage of each existing methods, a new ontology model has been established for power

transformer fault diagnosis by using OWL and protégé, hoping which can provide a new perspective for transformer diagnosis. The application of this system enables the integration of the diagnosis techniques of TM, DGA, and FRA, thereby enhancing the diagnostic efficiency of fault phenomena, fault sources and causes of faults. This is an open system which includes several mainstream transformer fault diagnosis methods, and more diagnosis methods can be added to improve this system in terms of its efficiency and veracity of fault diagnosis.

Constantly update of the existing diagnostic methods as well as new diagnostic methods continue to be developed, the use of the expansibility of the characteristics of ontology been added to the existing model to make it a better application in the future. Two major improvements can be made for this ontology diagnosis system.

- **Design a professional user interface so that non-expert can use this system.**
- **Try to add more transformer fault diagnosis methods into the ontology diagnosis model.**

Appendix A

Establishing Interface

Two softwares named Eclipse and Jena. Eclipse has been introduced in the main text and Jena is an open source Semantic Web framework for Java. It provides an API to extract data from and write to RDF graphs. The graphs are represented as an abstract model. A model can be sourced with data from files, databases, URLs or a combination of these.

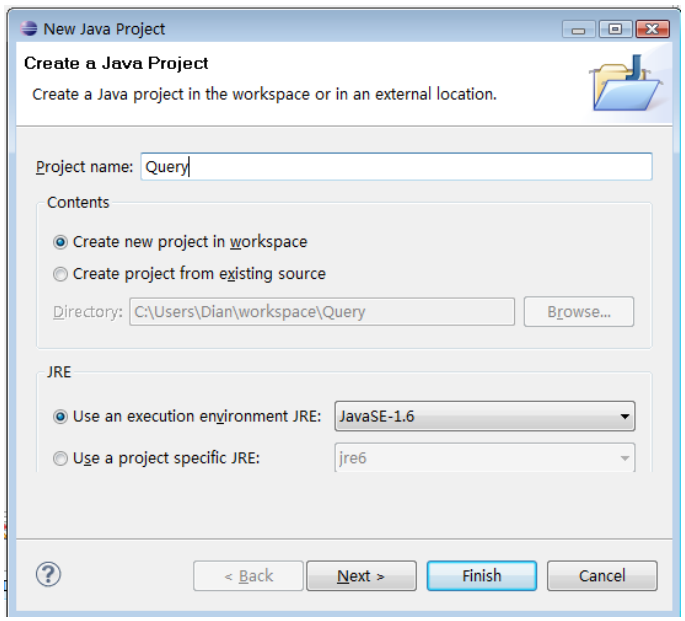

Figure A.1: Create a Java Project.

Firstly, build a new java project with eclipse, named **Query**. Shown as Figure A.1.

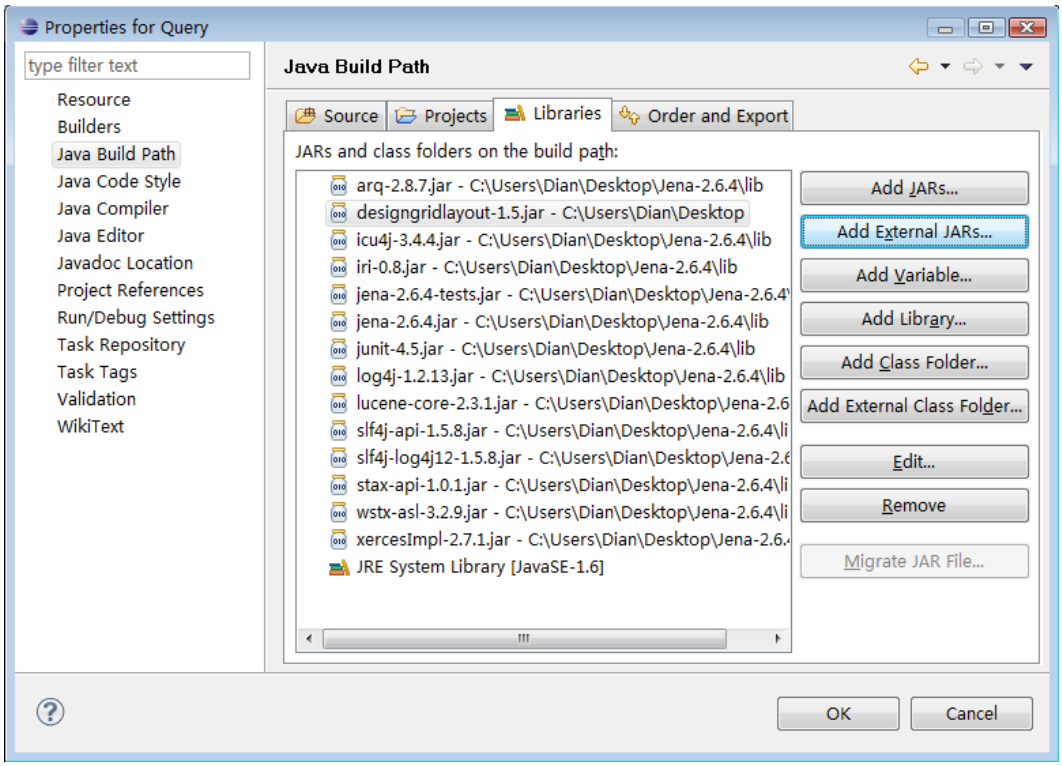

Figure A.2: Set the Java Building Path.

Adding **designgridlayout.jar** and all **jar** documents under lib folder in Jena to libraries list of java build path.Shown as Figure A.2.

Building a folder named **lib** under a java project named **Query**. Then, add **designgridlayout.jar** and all **jar** documents under lib folder in Jena to folder named **lib** which is under java project **Query**. Meanwhile, add transformer fault diagnosis ontology **Final.owl** built with Protégé to java project **Query**. Shown as Figure A.3.

Within the above mentioned procedures, all the background settings of Eclipse has been finished. The following procedure is to build a new java class named **query** and build the interface program with Eclipse under this class. Shown as Figure A.4.

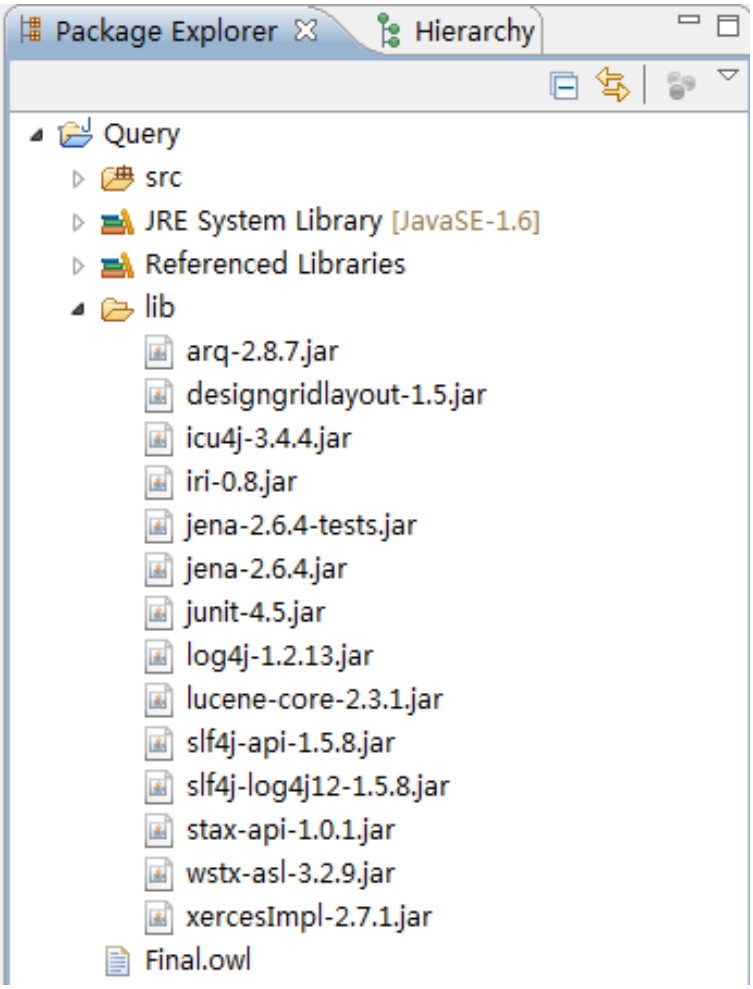

Figure A.3: Set the background.

Finally, run the program, the interface appears on the screen.

The details of the program are listed as follows:

```
package Faultdiagnosis;

import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.FileInputStream;
```

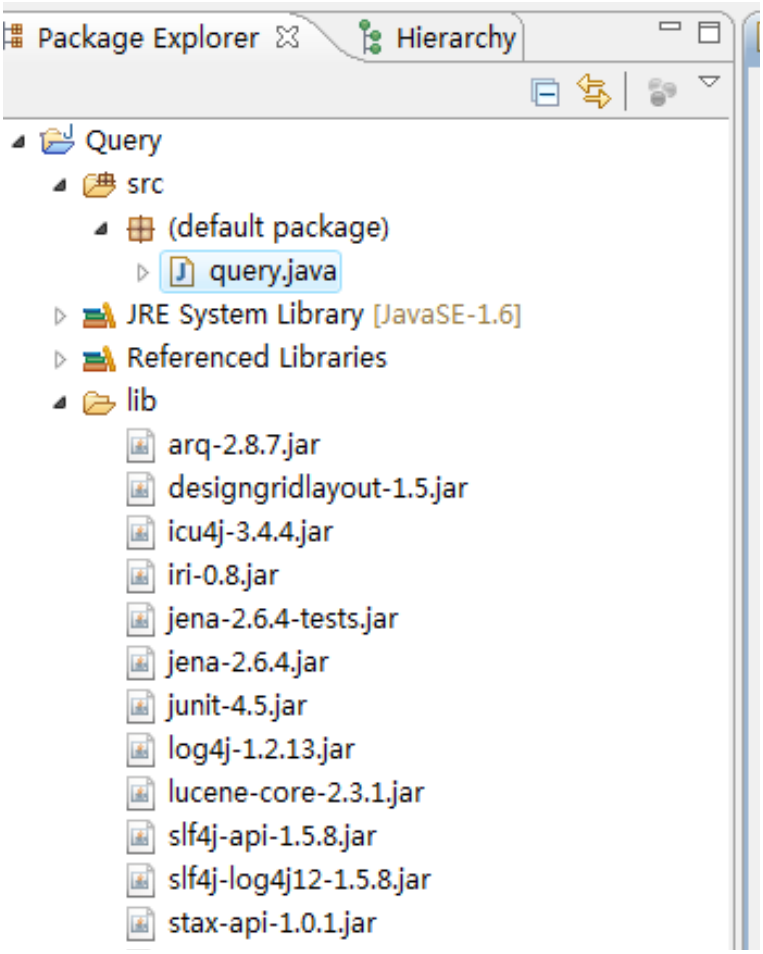

Figure A.4: Add the query class.

```
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.JTextArea;
import javax.swing.UIManager;
```

```
import net.java.dev.designgridlayout.DesignGridLayout;

import com.hp.hpl.jena.ontology.Individual;
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.ontology.OntResource;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.NodeIterator;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;

public class query {

private static JFrame frame;
private JComboBox FP;
    private JButton Query;
private JTextArea FR;
private JScrollPane FRScrollPane;
private JTextArea RS;
private JScrollPane RSScrollPane;
private OntModel model;

public query() {

initFrame();
initContentPane();
initActions();
loadOntology();
frame.setVisible(true);
}
```

```
private void initFrame() {
frame = new JFrame(
"Ontology Model for Power Transformer Fault Diagnosis");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(600, 400);
double width = Toolkit.getDefaultToolkit().getScreenSize().
                        getWidth();
double height = Toolkit.getDefaultToolkit().getScreenSize().
                         getHeight();
frame.setLocation((int) (width - frame.getWidth()) / 2,
(int) (height - frame.getHeight()) / 2);
frame.setResizable(false);
}


private void initContentPane() {
FP = new JComboBox();
Query = new JButton("Query");
Query.setEnabled(false);

FR = new JTextArea();
FR.setEditable(false);
FRScrollPane = new JScrollPane(FR);

RS = new JTextArea();
RS.setEditable(false);
RSScrollPane = new JScrollPane(RS);

DesignGridLayout layout = new DesignGridLayout
                                  (frame.getContentPane());
```

```
layout.row().grid(new JLabel("Fault Phenomenon:")).add(FP, 4)
.add(Query);

layout.row().center().fill().add(new JSeparator());
layout.row().grid(new JLabel("Fault Reason:")).add(FRScrollPane);

layout.row().center().fill().add(new JSeparator());
layout.row().grid(new JLabel("Repair Suggestion:")).
                add(RSScrollPane);
}

private void initActions() {

Query.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {

FR.setText("");
RS.setText("");

Individual fp = model
.getIndividual("http://www.owl-ontologies.com/unnamed.owl#"
+ FP.getSelectedItem());

NodeIterator p2r = fp
.listPropertyValues(model
.getObjectProperty
("http://www.owl-ontologies.com/unnamed.owl#Phenomenon_to_Reason"));
boolean first = true;
while (p2r.hasNext()) {
```

```
if (!first){
FR.append("\n");
}
Resource fr = (Resource) p2r.nextNode();
FR.append(fr.getLocalName());
first =false;
}

NodeIterator p2rs = fp
.listPropertyValues(model
.getObjectProperty
("http://www.owl-ontologies.com/unnamed.owl#Phenomenon_to_Repair"));

first = true;
while (p2rs.hasNext()) {

if (!first){
RS.append("\n");
}
Resource rs = (Resource) p2rs.nextNode();
RS.append(rs.getLocalName());
first =false;
}

}
});

FP.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
if (FP.getSelectedItem()!=null){
Query.setEnabled(true);
```

```
}else {
Query.setEnabled(false);
}
}
});

}

private void loadOntology() {
model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
try {
model.read(new FileInputStream("Final.owl"), null);
} catch (Exception ex) {

ex.printStackTrace();
JOptionPane.showMessageDialog(frame, ex.getMessage(),
"Loading Ontology Error!", JOptionPane.ERROR_MESSAGE);
System.exit(0);
}

OntClass FPClass = model
.getOntClass
        ("http://www.owl-ontologies.com/unnamed.owl#Fault_Phenomenon");

ExtendedIterator<extends OntResource> it = FPClass.listInstances();

while (it.hasNext()) {
OntResource fp = it.next();
FP.addItem(fp.getLocalName());
}
FP.setSelectedItem(null);
```

```
}


public static void main(String[] args) {
try {
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
} catch (Exception e) {
JOptionPane.showMessageDialog(frame, "UI Error!", "UI Error!",
JOptionPane.ERROR_MESSAGE);
}

new query();
}
}
```

Bibliography

[1] J.H. Provanzana and P.R. Gattens. Transformer condition monitoring realizing an integrated adaptive analysis system. In *CIGRE 1992 session*. 1992.

[2] W. Knorr T. Leibfried and K. Viereck. On-line monitoring of power transformers - trends, new development and first experiences. In *CIGRE 1998 session*. 1998.

[3] I.J. Kemp. Partial discharge plant-monitoring technology: present and future developments. In IEEE, editor, *Measurement and Technology*, pages 142:4–10. IEE Proceedings - Science, Measurement and Techonolgy, 1995.

[4] L. James et al. Model-based monitoring of transformers. Technical report, Massachusetts Institute of Technology, Laboratory for Electromagnetic and Electronic Systems, 1995.

[5] Iec60354: Loading guide for oil-immersed power transformers. Technical report, International Electrotechnical Commission Standard, Geneva, Switzerland, 1991.

[6] Iec60905: Loading guide for dry-type power transformers. Technical report, International Electrotechnical Commission Standard, Geneva, Switzerland, 1987.

[7] Transformers committee of the ieee power engineering society. ieee guide to loading mineral oil-immersed transformer, ieee std c57.91-1995. Technical

report, The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, U.S.A., 1991.

[8] Guide for loading oil-immersed distribution and power transformers. Technical report, American National Standards Institute, 1981.

[9] Guide for loading oil-immersed power transformers with 65 average winding rise. Technical report, National Electrical Manufacturers Association, 1978.

[10] Z. Radakovic and D.J. Kalic. Results of a novel algorithm for the calculation of the characteristic temperatures in power oil transformers. In *Electrical Engineering 80*, pages 205–214. 1997.

[11] Interpretation of the analysis of gases in transformers and other oil-

lled electrical equipment in service. Technical report, International Electrotechnical Commission Standard, Geneva, Switzerland, 1978.

[12] Transformers committee of the ieee power engineering society. ieee guide for the interpretation of gases generated in oil immersed transformers, ieee std c57.104-1991. Technical report, The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY, 10017, U.S.A., 1994.

[13] A. Mollmann and B. Pahlavanpour. New guidelines for interpretation of dissolved gas analysis in oil-filled transformers. In *Electra, CIGRE France*, pages 186:30–51. October 1999.

[14] Guide for the dissolved gas analysis and diagnosis. Technical report, Chinese standards boards, P.R.China, 1987.

[15] Y. Zhang Y.L. Liu, P.J. Griffin and X. Ding. An artificial neural network approach to transformer fault diagnosis. In *IEEE Transaction on Power Delivery*, pages 4:1838–1841. October 1996.

[16] Z.Y. Wang P.J. Griffin and Y.L. Liu. A combined ann and expert system tool for transformer fault diagnosis. In *IEEE Transaction on Power Delivery*, pages 4:1224–1229. October 1998.

[17] J.M. Ling C.E. Lin and C.L. Huang. An expert system for transformer fault diagnosis and maintenance using dissolved gas analysis. In *EEE Transaction on Power Delivery*, pages 1:231–238. 1993.

[18] T. Wu S.M. Islam and G. Ledwich. A novel fuzzy logic approach to transformer fault diagnosis. In *IEEE Transaction on Dielectrics and Electrical Insulation*, pages 2:177–186. April 2000.

[19] R.R. Rogers. Ieee and iec codes to interpret incipient faults in transformers using gas in oil analysis. In *IEEE Transactions on Electrical Insulation*, pages 5:348–354, October 1978.

[20] Iec60270: High-voltage test techniques-partial discharge measurements. Technical report, International Electrotechnical Commission Standard, Geneva, Switzerland, 2000.

[21] T. Bengtsson L. Ming, B. Jonsson and M. Leijon. Directivity of acoustic signals from partial discharges in oil. In *IEE proceedings - Science, Measurement and Technology*, pages 1:85–88. January 1995.

[22] A.K. Lazarevich. *Partial discharge detection and localization in high voltage transformers using an optical acoustic sensor*. PhD thesis, The Virginia Polytechnic Institute and State University, Blacksburg, Virginia, U.S.A., 2003.

[23] F.H.Kreuger E. Gulski and A. Krivda. Classification of partial discharges. In *IEEE transactions on Electrical Insulation*, pages 6:917–940. December 1993.

[24] E. Gulski. Discharge pattern recognition in high voltage equipment. In *IEE proceedings - Science, Measurement and Technology*, pages 1:51–61, January 1995.

[25] F.H. Kreuger and E. Gulski. Computer-aided recognition of discharge sources. In *IEEE Transactions on Electrical Insulation*, pages 1:82–92. February 1992.

[26] S.S. Tsai. Power transformer partial discharge (pd) acoustic signal detection using fiber sensors and wavelet analysis, modeling, and simulation. Master's thesis, The Virginia Polytechnic Institute and State University, Blacksburg, Virginia, U.S.A., 2002.

[27] C. Wang X. Jin T.C. Cheng X. Dong S. Gao W. Jing A.S. Farag, M.H. Shewhdi and Z. Wang. On-line partial discharge calibration and monitoring for power transformers. In *Electric Power Systems Research*, pages 50:47–54. 1999.

[28] D.W. Gross and M. Soller. Partial discharge diagnosis on large power transformers. In *In Conference Record of the 2004 IEEE International-Symposium on electrical insulation*, pages 186–191. 2004.

[29] R.A. Salas C. Gonzalez, J. Pleite and J. Vazquez. Transformer diagnosis approach using frequency response analysis method. In *In IECON 2006 - 32nd Annual Conference on IEEE industrial electronics*, pages 2465–2470. 2006.

[30] N. Guarino and C. Welty. A formal ontology of properties, knowledge engineering and knowledge management: Methods, models and tools. *In 12th International Conference (EKAW 2000)*, pages 97–112.

[31] http://www.w3.org/tr/owl-features.

[32] http://plato.stanford.edu/entries/logic-ontology/.

[33] J.Q. Feng, J.S. Smith, and Q.H. Wu. Condition assessment of power system apparatuses using ontology systems. In *Proc. of IEEE/PES Transmission and Distribution Conference Exhibition: Asia and Pacific.* Dalian, China, 2005.

[34] M. Wooldridge and N.R. Jennings. *Intelligent Agents-Theories, Architectures, and Languages.* ISBN 3-540-58855-8. 1995.

[35] P. Norvig and S.J. Russell. *Artificial Intelligence: A Modern Approach.* Third edition edition, 2003.

[36] J.Q. Feng, Q.H. Wu, and J. Fitch. An ontology for knowledge representation in power systems. In *Proc. of IEE Control 2004*. University of Bath, UK, September 2004.

[37] http://www.w3.org/tr/owl-guide/.

[38] P. Jackson. *Introduction To Expert Systems.* ISBN 978-0201876864. 1998.

[39] J.C. Giarratano and G.D. Riley. *Expert Systems: Principles and Programming.* ISBN 978-0534384470. 2004.

[40] N.F. Noy and D.L. McGuinness. Ontology development: A guide to creating your first ontology. Technical report, 2001.

[41] G. Moulton A. Rector R. Stevens M. Horridge, S. Jupp and C. Wroe. A practical guide to building owl ontologies using protege. Technical report, The University of Manchester, 2007.