

# Developing a Computational Framework for Explanation Generation in Knowledge-based Systems and its Application in Automated Feature Recognition

A thesis submitted in fulfilment of the  
requirements for the degree of  
Doctor of Philosophy

*Qingmai Wang*

B.Sc., M.Eng.

School of Electrical and Computer Engineering  
College of Science, Engineering and Health  
RMIT University

*May 2012*

## Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; and, any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

Signature:

Qingmai Wang

School of Electrical and Computer Engineering

RMIT University

Melbourne, VIC 3001, Australia.

May 2012

## Acknowledgement

Foremost, I would like to express my sincere gratitude to my primary supervisor Prof. Xinghuo Yu for the continuous support of my Ph.D study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research. Without him this thesis would not have been possible.

I also would like to thank my second supervisor Dr. Wei Peng, whose advices and insights were invaluable to me, especially during the inception phase of the project.

In addition, I am grateful to the Cooperative Research Centre for Advanced Automotive Technology (AutoCRC) for funding this research, and for their useful training sessions which provided me many valuable information regarding how to conduct research.

Thanks also go to my friends and colleagues, Dr. Christian Van der Velden, Dr. Ajendra Dwivedi, Dr. Xiangjun Li, and Mr. Miguel Combariza, for their kind suggestions and helps.

Finally I really appreciate the continuous support and understanding from my loving family, especially my mom Maijun and my wife Yajun.

# Contents

<b>Declaration</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>List of Tables</b>	<b>x</b>
<b>List of Figures</b>	<b>xi</b>
<b>Publications</b>	<b>xiv</b>
<b>Abstract</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Explanation and Knowledge-Based Systems . . . . .	1
1.2.1 Explanation . . . . .	2
1.2.2 Knowledge-Based Systems . . . . .	4
1.2.3 Explanation in Knowledge Based Systems . . . . .	5
1.3 Motivations . . . . .	7
1.3.1 Integrating “Explaining Available Evidences” . . . . .	7



1.3.2	Reconstructive Explanations . . . . .	8
1.4	Research Objectives and Research Questions . . . . .	10
1.4.1	Research Objectives . . . . .	10
1.4.2	Research Questions . . . . .	11
1.5	Contributions . . . . .	12
1.6	Thesis Structure . . . . .	14
1.7	Summary . . . . .	16
<b>2</b>	<b>Literature Review</b>	<b>17</b>
2.1	Introduction . . . . .	17
2.2	Theories of Explanation . . . . .	18
2.2.1	Deductive-Nomological (DN) Model . . . . .	18
2.2.2	The Pragmatic Theory of Explanation . . . . .	20
2.2.3	Explanation in Cognitive Science . . . . .	21
2.3	Explanation Models for KBSs . . . . .	24
2.3.1	Methods for Generating Basic Content . . . . .	24
2.3.2	Users and Contexts Oriented Explanations . . . . .	29
2.4	Models for “Explaining Available Evidences” . . . . .	34
2.4.1	Rule-based Models . . . . .	34
2.4.2	Probabilistic Models . . . . .	36
2.4.3	Neural Network Models . . . . .	37
2.5	Summary . . . . .	38
<b>3</b>	<b>Explanation Framework</b>	<b>40</b>
3.1	Introduction . . . . .	40

3.2	Preliminaries . . . . .	41
3.3	General Structure of the Framework . . . . .	42
3.4	Explanative Knowledge Base (EKB) . . . . .	44
3.4.1	OWL Ontology . . . . .	45
3.4.2	SWRL Rules . . . . .	49
3.4.3	Capturing Domain Knowledge . . . . .	50
3.5	Decision Explanation Model (DEM) . . . . .	52
3.5.1	Existing OWL Reasoners . . . . .	53
3.5.2	Developing a Novel Reasoner . . . . .	55
3.6	Software Diagnostic Model . . . . .	57
3.6.1	Abductive Reasoning . . . . .	58
3.6.2	Involving Interaction . . . . .	59
3.7	Summary . . . . .	61
<b>4</b>	<b>Backward Chained ABox Reasoner</b>	<b>63</b>
4.1	Introduction . . . . .	63
4.2	Preliminaries . . . . .	64
4.3	Concept Framework . . . . .	65
4.4	Building Rule Base . . . . .	66
4.5	Reasoning Algorithm . . . . .	67
4.5.1	Solution . . . . .	69
4.5.2	Searching for Solutions . . . . .	69
4.5.3	Generating Reasoning Results . . . . .	71
4.5.4	Backward Chaining . . . . .	73

4.6	Handling Special Rules and Atoms . . . . .	73
4.7	Experiments . . . . .	75
4.7.1	Test Ontologies and Queries . . . . .	76
4.7.2	Experimental Results . . . . .	78
4.8	Summary . . . . .	79
<b>5</b>	<b>Multiple Run Interactive Certainty Network</b>	<b>81</b>
5.1	Introduction . . . . .	81
5.2	Abductive Reasoning with Interactions . . . . .	82
5.3	Certainty Network Construction . . . . .	83
5.4	An Algorithm for Interactive Behavior . . . . .	86
5.5	General Interactive Reasoning Process . . . . .	88
5.6	Case Study . . . . .	89
5.7	Summary . . . . .	94
<b>6</b>	<b>Automated Feature Recognition</b>	<b>96</b>
6.1	Introduction . . . . .	96
6.2	Overview of Automated Feature Recognition (AFR) . . . . .	97
6.2.1	Why AFR? . . . . .	97
6.2.2	Major AFR Methods . . . . .	100
6.3	Boundary Representation of Solid Models . . . . .	102
6.4	Standard for the Exchange Product Model Data (STEP) . . . . .	104
6.4.1	Contents of STEP . . . . .	105
6.4.2	EXPRESS . . . . .	107
6.5	Existing AFR Systems . . . . .	108

6.6	Summary . . . . .	111
<b>7</b>	<b>Ontology-based AFR Explanatory System</b>	<b>112</b>
7.1	Introduction . . . . .	112
7.2	Overview of OAES . . . . .	113
7.3	Building Explanative Knowledge Base . . . . .	116
7.3.1	Capturing EXPRESS Schema of STEP AP203 . . . . .	116
7.3.2	Defining Feature Recognition Rules . . . . .	117
7.3.3	Inserting Additional Descriptive Information . . . . .	124
7.3.4	Inputting STEP Instances . . . . .	125
7.4	Decision Explanation Model . . . . .	127
7.4.1	Generating Decision Explanations . . . . .	127
7.4.2	Demonstration Example . . . . .	129
7.5	Software Diagnostic Model . . . . .	132
7.6	Usage of OAES . . . . .	135
7.7	Summary . . . . .	145
<b>8</b>	<b>Conclusion and Future Studies</b>	<b>147</b>
8.1	Introduction . . . . .	147
8.2	Conclusions . . . . .	147
8.3	Limitations and Future Works . . . . .	150
8.4	Summary . . . . .	153
	<b>Bibliography</b>	<b>154</b>

<b>A</b>	<b>Recognition rules in OAES</b>	<b>174</b>
A.1	Pocket Feature Family . . . . .	174
A.2	Hole Feature Family . . . . .	177
A.3	Blind Step Feature Family . . . . .	179
A.4	Through Step Feature Family . . . . .	181
A.5	Blind Slot Feature Family . . . . .	182
A.6	Through Slot Feature Family . . . . .	184
A.7	Basic constructs . . . . .	187

# List of Tables

1.1	Comparison between traditional computer-based information systems and KBSs . . . . .	4
3.1	Syntax and semantics of OWL constructs and axioms . . . . .	48
3.2	Syntax and semantics of SWRL atoms . . . . .	50
4.1	Translating ECA to SWRL-like rules . . . . .	67
4.2	Details of test ontologies . . . . .	76
5.1	Empirical data adopted from an automobile fault diagnostic scenario	91
7.1	Mapping from EXPRESS to OWL . . . . .	117
7.2	Mapping between OWL and EXPRESS items within the <i>triangular_pocket</i> example . . . . .	123
7.3	Probabilistic causalities with conditional probabilities . . . . .	133

# List of Figures

2.1	Explanation equation of DN model . . . . .	18
3.1	General structure of the explanation structure . . . . .	43
3.2	An example of OWL ontology . . . . .	47
3.3	Capturing domain knowledge using OWL and SWRL . . . . .	51
3.4	BCAR in generating decision explanations . . . . .	56
3.5	The general process of interactive AR . . . . .	60
4.1	Concept framework of BCAR . . . . .	65
4.2	The root of searching tree . . . . .	70
4.3	Generating branches . . . . .	71
4.4	Searching tree in BCAR . . . . .	72
4.5	Experimental results . . . . .	78
5.1	Causal network . . . . .	85
5.2	Interactive reasoning process . . . . .	88
5.3	Instantiated certainty network . . . . .	90
5.4	The functional process of MRICN . . . . .	92
5.5	The inference flow . . . . .	94

6.1	CNC machines . . . . .	98
6.2	Concept framework of the GA-based intelligent CAD/CAM system .	99
6.3	CAPP-based design cycle for metallic aerospace components . . . . .	100
6.4	Face-surface, edge-curve, and vertex-point . . . . .	103
6.5	Basic data structure of B-rep . . . . .	104
7.1	Ontology-based AFR explanatory system . . . . .	115
7.2	Portion of subtype entity hierarchy of <i>representation_item</i> . . . . .	118
7.3	Example of a triangular pocket . . . . .	119
7.4	Explainable features in OAES . . . . .	120
7.5	Descriptive information of <i>edge</i> . . . . .	125
7.6	Descriptive information of <i>concave_angular_edge</i> . . . . .	126
7.7	A sequence of explanations for “why <i>F001</i> is a <i>triangular_pocket</i> ” . .	130
7.8	Probabilistic causality network for software diagnosis in OAES . . . .	132
7.9	The dialog history in the software diagnostic model . . . . .	134
7.10	Internal structure of the F-35 strike fighter . . . . .	135
7.11	The main interface of OAES . . . . .	136
7.12	Ontology class tab . . . . .	137
7.13	Ontology property tab . . . . .	137
7.14	The SWRL rule tab . . . . .	138
7.15	Class detail panel for <i>face_bound</i> . . . . .	138
7.16	Explanation & query panel . . . . .	139
7.17	The explanation panel for <i>obround_hole</i> feature . . . . .	140
7.18	The explanation panel for <i>obround_hole</i> feature cont’d . . . . .	141



7.19	The explanation panel for <i>triangular_pocket</i> feature . . . . .	142
7.20	The explanation panel for <i>regular_through_slot</i> feature . . . . .	142
7.21	The query panel for <i>concave_angular_edge</i> . . . . .	143
7.22	Interface of the software diagnostic model . . . . .	144
7.23	Interface of the software diagnostic model cont'd . . . . .	144

# Publications

W. Peng, Q. Wang, B. Wang, X. Yu, “Putting simple hierarchy into ant foraging: cluster-based soft-bots”, in *Proceedings of the Third International Conference on Network and System Security (NSS’09)*, pages 484-488, Gold Coast, Australia, 2009

Q. Wang, W. Peng and X. Yu, “Ontology based geometry recognition system for STEP”, in *Proceedings of the 2010 IEEE International Symposium on Industrial Electronics (ISIE 2010)*, pages 1686-1691, Bari, Italy, 2010

Q. Wang, W. Peng and X. Yu, “Building an explanation generation mechanism in probabilistic knowledge-based systems”, in *Proceedings of the Eighth IEEE International Conference on Industrial Informatics (INDIN 2010)*, pages 229-233, Osaka, Japan, 2010

Q. Wang and X. Yu, “Improving reasoning capabilities for ontology-based geometric product model”, in *Proceedings of the 37th Annual Conference of IEEE Industrial Electronics Society (IECON 2011)*, pages 3048-3053, Melbourne, Australia, 2011

Q. Wang and X. Yu, “Reasoning over OWL/SWRL ontologies under CWA and UNA for industrial applications”, in *Proceedings of the 24th international conference on Advances in Artificial Intelligence (AI’11)*, pages 789-798, Perth, Australia, 2011

## Abstract

A Knowledge-Based System (KBS) is essentially an intelligent computer system which explicitly or tacitly possesses a knowledge repository that helps the system solve problems. Researches focusing on building KBSs for industrial applications to improve design quality and shorten research cycle are increasingly attracting interests. For the early models, explainability is considered as one of the major benefits of using KBSs since that most of them are generally rule-based systems and the explanation can be generated based on the rule traces of the reasoning behaviors.

With the development of KBS, the definition of knowledge base is becoming much more general than just using rules, and the techniques used to solve problems in KBS are far more than just rule-based reasoning. Many Artificial Intelligence (AI) techniques are introduced, such as neural network, genetic algorithm, etc. The effectiveness and efficiency of KBS are thus improved. However, as a trade-off, the explainability of KBS is weakened. More and more KBSs are conceived as black-box systems that do not run transparently to users, resulting in loss of trusts for the KBSs. Developing an explanation model for modern KBSs has a positive impact on user acceptance of the KBSs and the advices they provided.

This thesis proposes a novel computational framework for explanation generation in KBS. Different with existing models which are usually built inside a KBS and generate explanations based on the actual decision making process, the explanation model in our framework stands outside the KBS and attempts to generate explanations through the production of an alternative justification that is unrelated to the actual decision making process used by the system. In this case, the knowledge and

reasoning approaches in the explanation model can be optimized specially for explanation generation. The quality of explanation is thus improved. Another contribution in this study is that the system aims to cover three types of explanations (where most of the existing models only focus on the first two): 1) decision explanation, which helps users understand how a KBS reached its conclusion; 2) domain explanation, which provides detailed descriptions of the concepts and relationships within the domain; 3) software diagnostic, which diagnoses user observations of unexpected behaviors of the system or some relevant domain phenomena.

The framework is demonstrated with a case of Automated Feature Recognition (AFR). The resulting explanatory system uses Semantic Web languages to implement an individual knowledge base only for explanatory purpose, and integrates a novel reasoning approach for generating explanations. The system is tested with industrial STEP files, and delivers good quality explanations for user queries about how a certain feature is recognized.

# Chapter 1

## Introduction

### 1.1 Introduction

This chapter discusses the backgrounds and motivations underpinning this research work. Research objectives and contributions are also proposed.

The rest of the chapter is organized as follows: Section 1.2 introduces the concepts of explanation and Knowledge-Based System (KBS), as well as the role of explanation in KBS; Section 1.3 discusses the new ideas that inspire and underpin this research; Section 1.4 proposes the research objectives and research questions; Section 1.5 highlights the main contributions; Section 1.6 shows the thesis structure; and finally Section 1.7 summarizes the chapter.

### 1.2 Explanation and Knowledge-Based Systems

This section briefly introduces the philosophical ideas of explanation and the definition of KBS, as well as the usage of explanation in KBS.

### 1.2.1 Explanation

Explanation, according to Oxford English Dictionary, is commonly regarded as a statement or account that makes something clear. However, until now there still is not any well-accepted theoretical model that perfectly defines explanation within the philosophy of science. Historically, explanation usually refers to the causation, namely to explain an event or phenomena is to find out what has caused it. Thus, for many years, Aristotle's theories of causation have been regarded as the foremost theories of explanation.

In the Twentieth Century, philosophers started to look for theoretical models for explanations. A model of explanation is a set of necessary and sufficient conditions that determine whether an explanation correctly explains the question [2]. The first formalized model, Deductive-Nomological (DN) model, which is also regarded as the most significant model, was proposed by Hempel in 1940s [50], highlighting that the occurrence of a phenomenon should be explained by showing that the phenomenon is resulted from certain facts in accordance with a set of general laws. The major dispute over DN model is that a DN explanation is not necessarily a cause for the phenomenon to be explained. This leads to two issues: Explanatory Asymmetries and Explanatory Irrelevancies [147]. Salmon's Statistical Relevance (SR) model [110] is a very influential attempt to address these issues in terms of the notion of statistical relevance or conditional dependence relationships. The intuition underlying the SR model is that statistically relevant relationships are explanatory, while statistically irrelevant relationships are not [147]. As some philosophers argued that causal relationships are greatly underdetermined by statistical relevance relations [81], Salmon

finally abandoned the attempts in modeling explanations or causal relationships completely by using statistical methods. Instead, he proposed a Causal Mechanical (CM) model [108], claiming that an explanation can be represented by the causal processes and interactions leading up to the event that need to be explained.

In contrast to the above theoretical approaches which attempt to model an explanation as a set of conditions, other researchers hold that such models actually do not exist [2]. It seems that, no matter what model is applied, people can always find counter examples that do not fit the model. Van Fraassen claimed in his pragmatic theory of explanation [34] that “an explanation is not the same as a proposition, or an argument, or a list of propositions; it is an answer to a why-question”. As a consequence, instead of modeling the cause of the phenomenon to be explained, he focused on capturing the context and intention of the why-question in order to evaluate explanations. A deeply pragmatic explanation theory, the Illocutionary Theory proposed by Achinstein [1], described explanation as an illocutionary act that is typically performed by uttering words in certain contexts with appropriate intentions [2]. According to Achinstein, S (a person) explains q (an interrogative expressing some question Q) by uttering u only if S utters u with the intention that his u render q understandable by producing the knowledge of the proposition expressed by u that it is a correct answer to Q [1].

Generally speaking, theories of explanation provide fundamental ideas and principles for generating explanations using machines. For example, the DN model gives a feasible format of explanation, that is, the facts that cause a certain event plus the rules that lead the facts to the event can be generally accepted as the explanation of the occurrence of the event. In addition, the pragmatic theory reminds us that

Table 1.1: Comparison between traditional computer-based information systems and KBSs

Traditional Computer-Based Information System	Knowledge-Based System
Gives a guaranteed solution and concentrates on efficiency	Adds power to the solution and concentrates on effectiveness without any guarantee of solution
Data and/or information processing approach	Knowledge and/or decision processing approach
Assist in activation related to decision making and routine transactions; supports need for information	Transfer of expertise; takes a decision based on knowledge, explains it, and upgrades it, if required
Manipulation method is numeric and algorithmic	Manipulation method is primarily symbolic or connectionist
These systems do not make mistakes	These systems learn by mistakes
Need complete information and/or data	Partial and uncertain information, data, or knowledge will do
Works for complex, integrated, and wide areas in a reactive manner	Works for narrow domains in a reactive and proactive manner

explanations should vary in different contexts or with different intensions.

## 1.2.2 Knowledge-Based Systems

A KBS is essentially a computer-based intelligent system which explicitly or tacitly possesses a knowledge repository that helps the system make decisions or solve problems. The knowledge is usually generated from data or information inputted by a user, and can be captured using various knowledge representation techniques, such as rules, frames or connectionist models. The systems are capable of understanding the information being processed and can make decisions based on it [3], whereas traditional computer systems do not understand the data/information they process. Table 1.1 [105] shows the differences between traditional computer-based information systems and KBSs.



Some of the objectives of KBSs are listed below [134]:

- Provide a high intelligence level;
- Assist people in discovering and developing unknown fields;
- Offer a vast amount of knowledge in different areas;
- Aid in management of knowledge stored in the knowledge base;
- Solve social problems in a better way than the traditional computer-based information systems;
- Acquire new perceptions by simulating unknown situations;
- Offer significant software productivity improvement;
- Significantly reduce cost and time to develop computerized systems;

### **1.2.3 Explanation in Knowledge Based Systems**

Modern KBSs integrate increasingly complex techniques and methodologies to solve problems. Due to the complexity of the techniques integrated, systems' problem solving behaviors and strategies are usually opaque and vague to the end user. Misunderstandings and ambiguities thus emerge and often result in users' distrust of the intelligent systems, finally decreasing the acceptance of the systems. Putting explanatory processes into KBSs proves to be a viable solution. Explanations help users understand the causality about system behavior so as to increase user acceptance. Beliefs towards KBSs can be increased not only by the quality of its output, but more importantly, by the evidence of how and where it was derived.

The importance of explanation in KBSs has long been realized. From the first expert system, MYCIN [114], to many other followers, explanatory facilities have been integrated to help users understand systems better. According to Chandrasekaran [24], the research of explanation for KBS includes three aspects: 1) generating explanation's basic content; 2) responsiveness; 3) the human-computer interface.

Generating the basic content of explanations refers to what explanation should be provided and how to generate explanations from a KBS. For the first issue, Lacave and Diez [75] summarized three basic explanation focuses in KBSs: 1) Explaining available evidences, which consists in determining which hypothesis justifies the observed findings; 2) Explaining models, which provides detailed information for elements, relationships, components or events with the aim of helping users understand the system; 3) Explaining the reasoning process, which provides justification for the results obtained by the system and reasoning process that produced them. For the second issue, Chandrasekaran [24] classified existing approaches into two major categories: 1) by introspecting: explanations are generated based on system's own problem solving activity and by picking appropriate traces of reasoning steps or knowledge base portions used in making the decision; 2) by concocting: also known as reconstructive explanations, which means that explanations are generated by producing a justification that does not relate to how the decision was actually made.

Responsiveness, also called adaption [75], fulfills the need of generating explanations for different users within different contexts. Cognitive studies of explanation patiently remind us that users may have different intentions when they ask for explanations. Capturing these intentions is critical to providing satisfactory explanations. A responsive (or adaptive) explanation model should be able to apply user goals,

states of knowledge, and the dialogue structure to filter, shape, and organize process output. This often requires User Modeling or Context Handler.

Researches regarding human-computer interface of explanatory facilities focus on the ways in which the explanation can be effectively presented and displayed to users. Which explanations are best presented in natural language and which in graphical form (e.g. flow chart) is one major problem that this function needs to deal with. Other issues include how a user should ask questions: by selecting items in a menu or by nature language? Are users allowed to interrupt system running and request an explanation?

## **1.3 Motivations**

This section discusses two major ideas motivating this study: integrating “explaining available evidences” and reconstructive explanations.

### **1.3.1 Integrating “Explaining Available Evidences”**

As previously discussed, three explanation focuses have been identified: “explaining available evidences”, “explaining models”, and “explaining the reasoning process”. In most cases, people only mention the last two focuses when discussing the idea of explanation in KBSs. Systems for explaining available evidences, which focus on finding the root cause for any observed symptoms or evidences, are not considered to be explanatory facilities in KBSs [75]. Instead, they are mostly utilized in medical diagnosis or industrial failure detection, where the generation of a hypothesis is the core activity of those systems.

In fact, if we carefully review several cognitive studies of explanation, it is common for users to have questions such as “what’s wrong?” or “what did I do?” [48]. One of the major findings in the survey in [37] is that “when a mismatch occurred between system performance and user expectation, the user may be made confused and lead to a decrease in trust”. We can then conclude that a user may need explanations while he/she is experiencing an unexpected behavior of the system. In this case, providing explanations becomes a form of diagnosing unexpected system behaviors, and the explanatory facility here is more like a software diagnostic tool. Explanation models for “explaining available evidences” can then be applied to address this issue. For example, once a user finds unexpected behaviors, such as halting, generating errors or warnings, or even collapsing, he or she can input these observations into the explanation model as evidences. The explanation model should be able to uncover the reasons for the observed evidences, present the reasons to user as explanations, and give appropriate solutions.

### **1.3.2 Reconstructive Explanations**

Traditionally, explanation models are mostly inside KBSs and generate explanations “by introspecting” (see Section 1.2.3). This type of explanation generation heavily relies on how the system obtains the results and how knowledge is represented in the system. For early rule-based expert systems in the last century, this is a relatively good approach. This is because the decision making process in such systems is simply rule-based reasoning, so that the internal logical sequence can be perfectly explained by tracing the rules that have been used. Many remarkable explanation models were

proposed in this line around 1980s and 1990s.

However, modern KBSs are becoming increasingly intelligent and may possess very complex problem-solving mechanisms or very abstract knowledge representation. They often involve hybrid computational algorithms, where human logic does not exist and explicit causality between input and output is hardly recognized. Generating satisfactory explanations “by introspecting” is difficult in this case. As it has been observed, explanations in today’s systems are “even poorer than the first expert system MYCIN” [75].

To address the above issue, we believe that the explanation model should be outside KBSs, and use the “by concocting” method for explanation reconstruction (see Section 1.2.3), which means generating explanations through the production of an alternative justification that is unrelated to the actual decision making process used by the system. The explanation model, in this case, may own an independent explanative knowledge base, and use this knowledge base to argue convincingly that the answer is correct without actually referring to the derivation process. Compared to the traditional way of generating explanations “by introspecting”, using the reconstructive method, the explanation model can be optimized independently so that more flexible and understandable explanations can be provided.

One of the major concerns is that reconstructive explanations may not be quite accurate, since that it is more like that the explanation model uses its own knowledge to “guess” how problems are solved and how decisions are made by the KBS. But this does not quite matter. In most times, perfectly accurate explanations are not necessary. Users are happy to accept plausible explanations that are simple, easy to understand and in everyday language, just as the popularization of science, or math-

emational proof which persuades without representing the exact process utilized by mathematicians. Metaphorically, a KBS can be seen as an expert who can effectively make decisions and solve problems, but is not good at explaining himself/herself, while the explanation model acts as a teacher or tutor whose knowledge may not be as expansive as the expert's, but has far superior communication skills, so that he/she can infer how the expert makes decisions based on his/her own knowledge and clearly explain the inference to people with different backgrounds.

## 1.4 Research Objectives and Research Questions

### 1.4.1 Research Objectives

In general, the objective of this Ph.D study is to draw ideas from Philosophy and Cognitive Science to develop an intelligent computational framework for explanation generation in KBSs. Based on the motivations discussed in Section 1.3, the explanation model inside the framework includes two key ideas:

1. The framework should cover all the explanation focuses. Namely it should be able to provide explanations for:
  - decisions that have been made by the KBS;
  - domain concepts and relationships;
  - system failures, such as software errors or warnings;
2. The explanation model in the framework should be outside the KBS, and generate explanations by producing a justification that does not relate to how the

decision was actually made. To this end, the model should:

- own a separate Explanative Knowledge Base (EKB);
- have an independent reasoning mechanism;
- be able to use its own knowledge to “guess” how the goal is achieved in the KBS;

### 1.4.2 Research Questions

Three major research questions have been identified to achieve the research objectives.

1. How to build the framework for the explanation model?

This question refers to a set of questions within the framework design for the explanation model, such as: What are the components included in the framework? How do they connect with each other to share the information? What is the information flow? How does the explanation model communicate with the KBS? How does the explanation model interact with users? and etc.

2. How to build EKB for the explanation model?

This question refers to how knowledge is encoded in EKB. Many techniques are available to represent knowledge, such as rule base, first order logic, bayesian network, ontology and etc. It is critical to choose an appropriate method to represent knowledge in EKB in the sense of understandability and descriptiveness. In addition, how to map the information from a particular domain to the selected knowledge model is another issue that should be addressed.

### 3. How to generate explanations from EKB?

Once EKB has been built, how to generate explanations then becomes a reasoning issue in EKB. For example, the explanation about how a KBS reached conclusions (output) is generated based on a knowledge track which logically connects the output and input of the KBS in EKB.

## 1.5 Contributions

Major contributions of the research to the body of knowledge are listed below:

1. The proposing framework firstly covers all the three explanation focuses. Traditional works only consider the models for the second focus “explaining models” and the third focus “explaining reasoning process” as the explanation facilities in KBS. However, as we discussed, the first focus “explaining available evidences”, is also integrated in our framework as a software diagnostic tool.
2. Differing from most of the existing explanation models, the explanation is generated using a reconstructive method in this study. The explanation model in the proposing framework benefits from generating more flexible and understandable explanations, since that knowledge representation and reasoning methods can be separately optimized for generating explanations in this case. Very few researches has been done along this line. The only available literature [143] was published about 20 years ago, with simple frame-based EKB, and only available for early rule-based expert systems.
3. Semantic Web languages, Web Ontology Language (OWL) and Semantic Web



Rule Language (SWRL) are introduced to build EKB for generating explanations. Semantic Web is motivated to represent web content in a form that is more easily machine-processable [7], namely allowing the computer to understand human language. Conversely, if the computer can understand what people speak, it is reasonable to believe they may speak what people understand. To this end, general methods for mapping the domain knowledge of a certain application to an OWL/SWRL ontology are developed.

4. A novel Backward Chained ABox Reasoner (BCAR) is developed for reasoning through EKB to generate explanations for decisions made by a KBS. Comparing with other general ontology reasoners, BCAR has advances in: 1) reasoning with closed world based information model; 2) handling SWRL rules; 3) producing explanations by using backward chaining;
5. A novel Multiple Run Interactive Certainty Network (MRICN) is developed to improve the capability of handling context in the diagnostic model. MRICN is a probabilistic reasoning network that provides interactive Abductive Reasoning (AR) for the purpose of explanation generation and diagnosis. The network can interact with users and draws new information to allow reflective searching for the optimal set of knowledge with the maximal certainty gain.
6. The explanation framework is implemented and demonstrated with a case of AFR as a novel Ontology-based AFR Explanatory System (OAES). With this particular case, OAES generates explanations for: 1) features that have been recognized; 2) geometrical concepts and relationships within the domain; 3) failures, errors or warnings that have been observed while using OAES. The

development of OAES includes two sub-contributions: 1) A STEP-OWL interpreter that maps STEP instances to ontology ABox; 2) A set of SWRL rules and OWL concepts describing design features;

## 1.6 Thesis Structure

This thesis is divided into seven chapters. This first chapter has provided an overview of explanation in both theoretical studies and KBSs. Research objectives, research questions and contributions of this study have also been addressed.

Chapter 2 reviews existing methods and technologies relevant to this study. Theories of explanation in Philosophy and Cognitive Science are introduced first as the foundation. Researches of explanation in KBS are then discussed according to the explanation aspects mentioned in Section 1.2.3. Existing studies for “explaining available evidences” are reviewed separately since that traditionally they are not considered as capable of generating explanations for KBSs.

Chapter 3 explains the framework for explanation generation. Several key issues are discussed first, followed by a detailed description of the framework. Further, three major components are discussed in a sequence:

- EKB: Technologies used for building EKB and the methods for mapping domain information to the EKB are described;
- Decision Explanation Model (DEM): Reasoning issues that were used to generate decision explanations are highlighted;
- Software Diagnostic Model (SDM): Key concepts within the diagnostic model

including AR and uncertainty handling are discussed;

Chapter 4 discusses the technical details of BCAR. The development of BCAR includes building a unified rule base, developing a reasoning algorithm, and handling some special logical atoms. A benchmark ontology is adopted to test the performance of BCAR.

Chapter 5 describes the technical details of MRICN. An introduction of AR with interactions is firstly given. Issues regarding how to construct the certainty network and how to implement the interactive reasoning process are then discussed. An experimental test is finally carried out.

Chapter 6 gives an introduction to the background of STEP-based AFR, which is used as the demonstration scenario for the explanation framework proposed in this thesis. The introduction includes: an overview of AFR principles, a brief description of Boundary Representation and STEP standard, and a review of existing AFR models.

Chapter 7 discusses the implementation and application of the explanation framework within AFR. A novel AFR Explanatory System, OAES, is proposed. OAES is developed to provide explanations for any AFR systems that recognize features from STEP files. The explanations produced are not related to how the features are actually recognized in the AFR systems. Instead, a set of recognition rules are constructed based on the face topology, and are used to generate explanations.

The final chapter concludes the thesis and proposes some future works.

## 1.7 Summary

This chapter has given a brief introduction of the concepts of explanation and KBS, as well as the usage of explanation in KBS. It is, then, followed by the discussions of motivations of the study, research objectives and questions, and contributions. Lastly, the thesis structure has been outlined.

# Chapter 2

## Literature Review

### 2.1 Introduction

This study covers multiple research disciplines, including Cognitive Science, Knowledge Engineering, Computer Science and Computer-aided Manufacturing design. This chapter generally discusses background knowledge and reviews existing methods and technologies in relation to this study.

The rest of this chapter is organized as follows: Section 2.2 reviews major theoretical ideas of explanation in both Philosophy and Cognitive Science; Section 2.3 discusses existing studies of explanations in KBS according to the three explanation aspects; Section 2.4 introduces previous studies for “explaining available evidences”; Section 2.5 summarizes the chapter.

## 2.2 Theories of Explanation

Several significant theoretical ideas mentioned in Section 1.2.1 are discussed in details in this section, as well as some results achieved from cognitive studies of explanation.

### 2.2.1 Deductive-Nomological (DN) Model

The DN model was proposed by Hempel [50] in 1940s, and was considered to be the first and most significant theoretical model of explanation.

According to Hempel, an explanation consists of two major constituents: an explanandum and an explanans. The explanandum describes the phenomenon to be explained and the class of those sentences that are adduced to account for the phenomenon is enclosed in the explanans. The explanans falls into two subclasses: one contains certain sentences  $C_1, C_2, \dots, C_k$  which state specific antecedent conditions; the other is a set of sentences  $L_1, L_2, \dots, L_r$  which represent general laws. The explanation equation is then defined as shown in Figure 2.1:

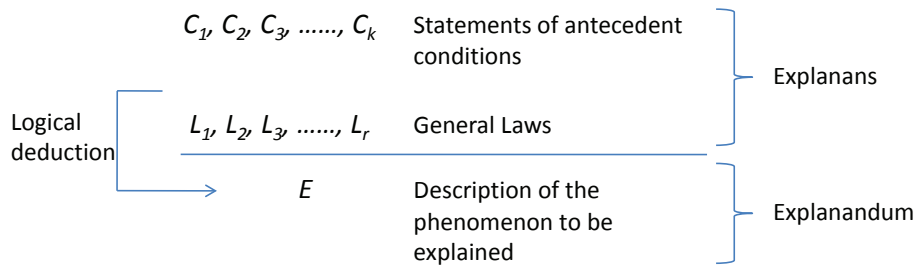


Figure 2.1: Explanation equation of DN model

Within this model, the constituents have to satisfy four conditions of adequacy being a sound explanation:

(R1) The explanandum must be a logical sequence of the explanans;

(R2) The explanans must contain general laws, and these must actually be required for the derivation of the explanandum;

(R3) The explanans must have empirical content;

(R4) The sentences constituting the explanans must be true;

One rather obvious problem in DN model has to do with the temporal relations between explanatory facts (singular sentences in the explanans) and the event to be explained (explanandum) [109]. For example, a total lunar eclipse can be satisfactorily explained by deducing its occurrence from the relative positions of the earth, sun, and moon at a certain time prior to the eclipse in conjunction with some laws of celestial mechanics. However, although it is equally possible to deduce the occurrence of the eclipse from the relative positions of the stars at some time after the eclipse in conjunction with the very same laws, hardly anyone would admit that the latter deduction qualifies as an explanation.

Another issue has to do with the role of causality in explanation. This includes cases where an event fits the model, but is not explained, and cases where the event is explained but does not fit the deductive structure of the model [47]. A classic example is that the length of a shadow can not explain the height of a flagpole, though the length of the shadow, the position of the sun, and the laws of geometry can be combined into an explanation to fit the DN model.

### 2.2.2 The Pragmatic Theory of Explanation

The pragmatic theory of explanation highlights the impact of the context of explanation. Van Fraassen [34] claimed that “an explanation is not the same as a proposition, or an argument, or a list of propositions; it is an answer to a why-question”. Consequently, instead of modeling the cause of the phenomena to be explained, the pragmatic theory focuses on capturing the context and intention of the questions in order to evaluate explanations. For example, he stated that both “because I flipped the switch” and “because we are expecting company” are explanations of the question “why is the light on?”, depending on the motivation for and the context of the original question.

Van Fraassen identified the abstract why-question with a triple:

$$Q = \langle P_k, X, R \rangle$$

where  $P_k$  is the topic of the question,  $X$  represents a contrast class which is a set of alternatives regarding the motivation of the question, and  $R$  denotes a relevance relation which determines what shall count as a possible explanatory factor.

More concretely, the contrast class provides information on why one particular event occurred instead of another in its contrast class, so that every why question “why  $x$ ?” may be translated as “why  $x$  instead of  $y$ ?”. For example, a question “why did Adam eat the apple?” may have three different intentions: (1) looking for an explanation which tells us why it was Adam (and not someone else) who ate the apple, (2) looking for an explanation which tells us why Adam ate (as opposed to doing something else to) the apple, and (3) wanting to know why Adam ate the apple (as opposed to Adam eating something other thing). The only difference between



these three intentions is a difference in contrast classes. In (1) the contrast class would consist in other people having eaten the apple. In (2), it would be Adam doing other things to the apple, and in (3), it would be Adam eating something other than the apple. After the contrast class is determined for a question, the best explanation can then be identified if it offers a larger probability to the questions' topic against other alternatives in the contrast class.

A problem in the above theory is that the relevance relation  $R$  has not been clearly defined. Van Fraassen only suggests that it is “determined by the context” in which the question is asked. Salmon and Kitcher [73] argue that according to Van Fraassen virtually anything can count as an answer to just about any why-question, because he does not place constraints on  $R$ .

### 2.2.3 Explanation in Cognitive Science

While explanation is essentially an act of communication, the behavior of explaining can be regarded as a purely cognitive activity, and an explanation can be considered as a certain kind of mental representation that results from or aids in this activity. Therefore, explanation has recently emerged as an important topic of study in both cognitive development and cognitive psychology ([70], [69]).

Cognitive Science is the interdisciplinary scientific study of mind and its processes. It integrates across multiple perspectives drawn from Biology, Psychology, Computer Science, and others, in order to understand how information is processed, represented, and transformed in behavior, nervous system or machine. In Cognitive Science, the primary research method is experimentation with human participants.

People, usually undergraduates satisfying course requirements, are brought into the laboratory so that different kinds of thinking can be studied under controlled conditions [127]. With these human participated experiments, studies of explanation in cognitive science usually help us understand explanations in an empirical way.

Gregor and Benbasaat [39] reviewed a group of empirical studies, mainly with KBSs, aiming to address several questions concerning the importance of explanations, such as “why are explanations needed?”, “what types of explanations should be provided?”, “when and how are explanations likely to be used?”. Nine propositions are then nominated with respect to several important factors of explanations, as listed below:

- Explanations will be used when a user experiences an expectation failure, or perceives an anomaly.
- Explanations will be used more when a user has a goal of long-term learning.
- Explanations will be used when a user lacks the knowledge needed.
- Explanations that require less cognitive effort to access and assimilate will be used more and will be more effective with respect to performance, learning, or user perceptions.
- Use of explanations improves the performance achieved with a KBS as an aid.
- Use of explanations helps in learning (transfer of knowledge to non-KBS contexts).
- Novices will use explanations more for learning.

- Experts will use explanations more for resolving anomalies.
- Use of explanations conforming to Toulmin’s model (justification explanations) will give rise to more positive user perceptions of a KBS than other explanations.

Intelligent agent is currently regarded as the most active topic in KBS. Therefore recent studies of explanation are motivated to improve user acceptance of those agents. [37] presents a survey of the testing users of CALO, a complex adaptive agent system, to investigate themes surrounding trust and understandability. Six themes regarding trustfulness were concluded based on participants’ feedback. For example, one theme that has been identified states that “many users commented that knowing what resources were being used to provide answers would aid them in trusting the system”.

In most of the existing works regarding explanations in KBS, explanation only refers to explaining decisions or actions made by the system itself. However, in a recent paper [48], the authors claim that the explanatory facility in modern intelligent agents should covers more, such as explanations of agent component or design rationale. For this reason, the authors classify the explanations into four categories: 1) ontological explanation: request for information regarding identity, definition, relation and event; 2) mechanistic explanation: request for information describing an agent’s behavior; 3) operational explanation: request for instructional content; 4) design rationale: request for information about design rationale. An experiment is then presented and the result is interesting. Actually the requests for mechanistic explanation only account for 19% of total requests, whereas the requests for ontological explanation account for 58%. This result reveals a broader way of thinking

about explanation facility in KBS. Future explanation models should not only explain “why?” and “how?” questions, but also attempt to explain “what?” questions.

## **2.3 Explanation Models for KBSs**

This section reviews existing studies in building explanatory facilities for KBSs according to the explanation aspects mentioned in Section 1.2.3. Methods for generating basic content of explanation are firstly discussed. A review of studies focusing on providing adapted explanations is followed. The third aspect, developing human-computer interface is not included since that it is only a software development issue. In addition, the explanation models discussed in this section only deal with the “explaining models” and “explaining reasoning process”. Models for “explaining available evidences” are reviewed separately in the next section, since that traditionally they are not considered as explanatory facilities in KBSs.

### **2.3.1 Methods for Generating Basic Content**

The significance of explanations was realized since the early expert system MYCIN [114]. MYCIN is a computer-based consultation system designed to assist physicians in the diagnosis of therapy advice for patients. It contains an explanation system, in which explanations are generated based on the rules that have been used or rules that are going to be used. Two components are involved in providing explanations [136]: 1) Question Answering (QA) program: QA answers simple English-language questions concerning the decisions made by the system in a particular consultation or the system’s knowledge in general. For example, to explain how the value of

a parameter is concluded, MYCIN retrieves the list of rules which were successfully applied, and presents them to the user along with the conclusions drawn. 2) Reasoning Status Checker: this component allows users to ask questions at any time during the consultation, and provides answers by checking current reasoning status. For example, when the user is asked a question, instead of answering the question he/she can ask why the question was asked. By checking current reasoning status, MYCIN simply answers that “because I’m trying to apply the following rule...”.

It has been argued that MYCIN has two major shortcomings [91]: firstly, there is no strategy of explanation about why a certain rule is used; secondly, deep explanations which justify the rules are not available in MYCIN. A series of the following models was developed to address these issues.

In contrast to MYCIN in which strategy knowledge (the diagnostic procedure) is implicitly embedded, NEOMYCIN [26], a successor of MYCIN, represents the strategy knowledge explicitly. NEOMYCIN’s strategy is structured in terms of tasks, which correspond to meta-level goals and subgoals, and meta-level rules (meta-rules) which are methods for achieving these goals [46]. Strategic explanations, which describe the plan the system is using to reach a solution, can then be generated by presenting the meta-rules to the user. A similar method, Generic Task approach [18] classifies the problem solving behaviors into 6 generic tasks. For example, one of the most important generic tasks is called hierarchical classification, which is to classify a situation description as one or more elements in the classification hierarchy. Once a KBS is built using the generic-task approach, the trace of the system’s problem solving behavior can be automatically represented at the architectural level in terms of its control strategy goals. Strategic explanations are naturally included

in the trace. Another task-based model that can provide strategic explanations is called CARMEN [132], which is a platform for building expert systems. CARMAN is built based on the Methodology of Modeling Control Knowledge (MMCK) which represents problem-solving entities explicitly and allows for flexible integration of different kinds of knowledge and reasoning strategies. MMCK divides a KBS into three layers: task layer, meta-knowledge layer and basic knowledge layer. Four types of explanation can be generated within this structure: 1) control decisions regarding planning tasks; 2) control decisions regarding decomposing a goal into sub-goals; 3) control decisions regarding evaluating the achievability of sub-goals; 4) how to apply domain knowledge to solve sub-problems.

XPLAIN ([121], [92]) is the first system that highlights the importance of deep explanation, a type of explanation that provides justification for the system behavior based on domain knowledge. In XPLAIN, the domain knowledge, containing the descriptive facts of the domain, such as causal relationships and classification hierarchies, and the domain principles (the procedural knowledge), containing the methods and heuristics of the domain, are separately captured. An automatic programmer integrates these prescriptive principles together with the descriptive facts of the domain to produce the performance program. This process of integration is recorded and used as the justification for the expert system’s behavior. Different from XPLAIN in which the deep knowledge is completely independent of the problem solving component, [23] proposed a complied knowledge approach that has all the relevant deep knowledge “complied” into the problem solving component. In such a way, the system can handle all the diagnostic problems that the deep knowledge is supposed to handle if it is explicitly represented and used in problem-solving.

Most of the explanation models generate explanations based on their own decision making process. An alternative approach is called reconstructive explanation approach ([143], [145] and [144]). The author argues that “A human expert, when asked to account for complex reasoning, rarely does so exclusively in terms of the actual process used to solve the problem. Instead, an expert tends to reconstruct a ‘story’ that accounts for the problem solving. This story reflects the expert’s line of explanation that is not necessarily the same as the original line of reasoning”. To this end, a Reconstructive Explainer (REX) was developed. REX is a test-bed system capable of producing reconstructive explanations for expert systems. The general idea of REX is to search for a restricted subset of an separate explanatory knowledge base in order to find a line of explanation supporting the KBS’s reasoning.

The authors of [52] believe that in-depth explanations are not necessary if the structure information and the functionality of the system are transparent to the user. They argue that “human explanations are based on the competence lacking in a computer system. A computer cannot create the insight that is required to the user to understand a specific aspect or decision. Instead, it can only communicate the necessary information. The real explanation is then created by the users themselves”. A four-layer explanation model is proposed, including a domain-layer (descriptive system-independent knowledge about the domain), a system-layer (general aspects of the system), a process-layer (dynamic behavior of the system) and a function-layer (functions accomplished by the system).

Recent studies in building explanation model are becoming diversified and isolated within different applications. One of the major branches focuses on explaining the increasingly popular intelligent agent. Debrief [61] is the first agent behavior ex-

planation model that was implemented in an artificial fighter pilot with the ability to explain the motivations for its actions, situation assessments, and beliefs. In Debrief, explanations are generated by “recalling the situation in which a decision was made”, “reconsidering the decision”, and “determining what factors were critical for the decision”. Another explainable agent is called Full Spectrum Command (FSC) [135] which is a training system developed for the U.S. Army by commercial game developers and academic researchers. FSC includes an Explainable AI (XAI) feature that allows the user to ask questions about the current behavior of the agent. XAI can extract key events and decision points from the agents so as to explain their behavior in response to the questions selected from the XAI menu. Harbers et al. [45] discussed that the XAI system provides information about an agent’s physical state and Debrief provides explanation in terms of an agent’s beliefs, and proposes a new model for explainable Belief-Desire-Intension (BDI) agents which enables the explanation of BDI agent behavior in terms of underlying beliefs and goals.

Another direction which has attracted a lot of interest is to explain recommendation systems. The recommendation systems capture user preferences in order to suggest items to assist users by offering relevant information on the web, and have been successfully implemented in many commercial web sites, such as Netflix and Amazon. The explanations, in this case, are often formulated as “Item A is recommended to you because ....” [129]. [131] has summarized seven design criteria for explanation facilities in recommendation systems, including: 1) Transparency: explaining how the system works; 2) Scrutability: allowing users to tell the system it is wrong; 3) Trust: increasing user confidence in the system; 4) Effectiveness: helping users make good decisions; 5) Persuasiveness: convincing users to try or buy; 6)



Efficiency: helping users make decisions faster; 7) Satisfaction: increasing the ease of usability or enjoyment. These design criteria can also be used for general explanation models, whereas the way in which an explanation can be measured for its effectiveness in each of the criteria remains an issue. Existing models for generating explanations for recommendations can be classified generally into three categories according to [139]: 1) item-based explanations: a set of intermediary items serves as the explanations by expressing the similarity between the items and the recommendations, as is implemented in the Netflix system <sup>1</sup>; 2) user-based explanations: explanation is provided by showing how other users with similar taste rated the recommended item [51]; 3) feature-based explanations: this type of approach uses features or characteristics of the recommended item as intermediary entities. For example, one movie recommender prototype uses movie features including genre, director, and cast to justify recommendations [130].

### 2.3.2 Users and Contexts Oriented Explanations

Explanation is not only a matter of giving access to the knowledge contained in the system. Pragmatic theories of explanation and many empirical studies suggest that explanations have to be adapted to the user's knowledge and responsive to the user's specific needs. A major part of research on explanation has been devoted to modeling user profile and customizing explanations, which adapts the form and content of explanations to the user's perspective and to cooperate with the user during problem solving.

The idea that computer systems would interact more effectively with users if they

---

<sup>1</sup><http://www.netflix.com>

had knowledge about their characteristics has received much attention in human-computer interaction. Modeling user profile refers to the process of gathering information about users and using the information to provide adaptive explanations. According to [83], user models can be classified along several dimensions:

- Individual vs. canonical: distinction between models for individual users and models for classes of users.
- Static vs. dynamic: dynamic models contain information that can change over time, while static models remain unchanged.
- Short term vs. long term: this feature refers to whether the user model information is discarded at the end of a session or is maintained for future usage.
- Explicit vs. implicit acquisition: user models can be defined through explicit acquisition during the user's interaction with the system or through implicit acquisition using inference methods.

User profile model has been used a lot to improve human-computer communication. A computer system, TAILOR [97], was proposed to generate adaptive descriptions of devices based on the user's level of knowledge which is captured in a user model. Two strategies were developed to construct a description for either a novice or an expert, and can be merged automatically to produce a wide variety of different descriptions for users who fall between the extremes of novice and expert. Another system [19] uses a user model that captures the user's plans and goals in order to deduce the intended meaning of the user's input so as to enhance the quality of

human-computer communication. Benyon and Murray [14] concluded a common architecture for applying user modeling to human-computer interaction design, which includes three sub-models: the user model, the domain model and the interaction model.

There are also many papers discussing the construction of the user profile model. A typical implicit user model acquisition approach, GUMAC [67], was developed in 1990s. GUMAC uses a set of heuristic rules to capture user beliefs from their interactions with an expert system, and is utilized by an explanation generator to tailor explanations the expert system gives to its users. Wu [148] argued that an effective user model acquisition system should actively ask the questions to acquire knowledge about the dialog partner, and he proposed an innovative architecture of dialog agent that can actively query users based on reasoning failures. Reusability was also discussed in order to reduce the cost of constructing and maintaining the user model [68]. Recent studies of user profile modeling tend to integrate technologies from many other disciplines. For example, fuzzy logic is introduced to describe the user's knowledge level [17] since that the author believes that "the description of users' knowledge level is an inaccurate one, so it is better to describe it in a fuzzy way instead of accurately". Ontological approaches [85] are also used to improve the user profiling performance

Beside user profile modeling, according to [87], the ability "to identify what context consists of and how each aspect of context can affect communication" is also important to "understand how a flexible explanation module should be designed and what knowledge sources it would be able to access". Based on this point of view, five elements of context were identified:

1. Problem solving situation: This refers to the problem solving situation in which the communication is taking place: what it is, at which point in the process the communication takes place, etc.
2. The participants: It is recognized that communication is affected by how much the participants know about the domain or the task under consideration.
3. The mode of interaction: The mode of interaction that is taking place also affects how it occurs, in particular, the medium used, whether feedback is allowed, and the number of participants.
4. The discourse: The discourse typically refers to the dialogue history and the current message, in terms of what is being communicated and how it was communicated and why.
5. The “external world”: Certain things do not change based on either the discourse or the problem solving state. For instance, different social situations often call for specific patterns of communication.

Context includes so many aspects that none of the existing research can cover all of them. They all focus on one or two aspects. In fact, context handling also includes user profile modeling, as the participants are regarded as one aspect of context as well. Generally, interactions between the user and the system are regarded as the major source of contextual information in most of the existing studies. They either actively request for more information or analyze the dialog history to identify the context. In the following discussion, several typical adaptive explanation models with context handling components are reviewed.

In an early command assistance system [104], examples are provided in explanations to offer a concrete illustration of what is being explained. The examples are tailored to be adapted to the context based on previous interaction. For example, if a user has just asked about “PRINT” and then asks what a “queue” is, the system would give examples of queues used for print jobs. Moore ([88], [89]) argued that previous explanation systems cannot “clarify misunderstood explanations, elaborate on previous explanations, or respond to follow-up questions in the context of the ongoing dialogue”, and a reactive approach to explanation was proposed. The approach firstly identifies a discourse goal based on the user’s query or the expert system, then heuristically searches for strategies to achieve the goal, taking into account the previous dialog, and finally plans the explanation and records the plan for further processing. An Explanatory Discourse GEnerator (EDGE) was proposed to generate explanations about electronic circuits ([21], [22]). The EDGE system combines goal-based reasoning, captured in content-planning rules, with dialogue conventions captured in dialogue-planning rules. Goal-based reasoning is used to identify and organize the content of an explanation, and dialogue conventions are used to manage the interaction and determine features such as the use of discourse makers and meta-comments. Another typical dialogue based interactive explanation system, P.rex, was developed to explain each proof step of a mathematical proof [32]. In P.rex, the user may interrupt the system at anytime whenever the explanation provided does not satisfy the user. The system then analyzes the dialog to uncover the reason, and re-plans a better adapted explanation.

## 2.4 Models for “Explaining Available Evidences”

“Explaining available evidences” refers to generating explanations for what has been observed, which are known as scientific explanations [128]. Systems for generating scientific explanations are often not regarded as explanation components for other KBSs. Instead, they are considered as independent KBSs themselves, which look for reasons for any observations or symptoms, e.g. MYCIN [114]. In this case, they can also be seen as a kind of diagnostic expert system [6]. The difference between diagnostic systems and scientific explanation generation is that diagnostic systems often cover many other sub-topics than just looking for reasons for observed symptoms, such as fault detection and system monitoring, while scientific explanation generation only focuses on explaining why things happen, which is one of the most important cognitive operations.

The core activity in “explaining available evidences” is called abduction (also called AR), which is a form of inference that generates a hypothesis best explaining the observation [63]. Several implementation models for AR are reviewed in the following discussions, including rule-based model, probabilistic model and neural network model.

### 2.4.1 Rule-based Models

Based on the classical deductive model, an explanation of a statement consists of a set of particular facts and a set of general rules [50]. Thus, deductive explanations often operate in rule-based systems in which the starting states, together with a set of rules, explain the goal state. In the rule-based model, a simple form of AR can

be modeled as a kind of backward chaining. Backward chaining starts from the goal state to find rules that could produce it from the starting state. It can also be seen as the process of generating hypotheses. Using backward chaining, it is possible to generate more than one competing hypotheses and users can accept only one of them as the explanation, so that additional process is required to select the best explanatory hypothesis and remove its competitors.

PI, which stands for “process of induction”, is a computational model that is able to perform AR as the inference to the best explanation ([122], [124]). To explain an observable data, the abductive inference integrated firstly generates a set of alternative hypotheses since the rules used for achieving the data may have multiple conditions (the observable data are regarded as the consequent of the rules). PI, then, evaluates the alternatives and selects the most appropriate hypothesis as the best explanation of the observable data, taking into account 3 criteria [128]: 1) concision, which is a measure of how much a hypothesis explains; 2) simplicity, which is a measure of how few additional assumptions a hypothesis needs to carry out an explanation; and 3) analogy, which favors hypotheses whose explanations are analogous to accepted ones.

There are many other similar works, except using different hypotheses evaluation and selection methods. For example, some researchers developed a cost-based (also called weight-based) abduction model ([25], [55]), in which they defined a “assignability cost” (real number) for each rule and each conjunct in the condition of the rule. This abduction model attempts to find the best explanation for a set of facts by finding a minimal cost hypothesis for the facts, where the costs are computed by summing the costs of assumption necessary for the hypothesis plus the cost for the rules.

### 2.4.2 Probabilistic Models

Bayesian Network (BN) is the dominant computational model for modeling explanation probabilistically ([98], [99]). In general, a BN is a directed acyclic graph, in which the nodes represent a set of random variables (often referring to the propositions in the explanation models) and the edges represent conditional dependencies between the random variables. BN is a convenient way for representing probabilistic causal relationships between propositions. In the case of using BN for representing causal relationships, a BN node represents a proposition which is a binary random variable taking “true” or “false” for values, indicating whether the proposition is true or false. On the other hand, a BN edge, directing from the cause to the consequent, represents the causal relationship between two propositions and carries a conditional probability implying how heavily the consequent depends on the cause. For example, a BN could be applied to represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Many powerful algorithms have been developed for making probabilistic inferences in BN and for generating explanations for some observed evidences. Existing approaches for generating explanations are mostly based on Maximum a Posteriori assignment (MAP) and Most Probable Explanation (MPE) [98]. MAP finds a complete instantiation of a set of potential hypotheses that maximizes the joint posterior probability, given partial evidence on the other variables, while MPE is similar to MAP, except that MPE defines the potential hypotheses to be all the unobserved variables. However, these approaches always return a configuration of all the poten-



tial hypotheses as the explanation without identifying which of them are important. Many other techniques have been introduced in order to find the most probable explanations for the given evidences, such as divide and conquer approach ([76], [93]), niching genetic algorithm [118] and junction tree algorithm [113]. Besides finding the most probable explanations, there are also some approaches that were developed to identify important hypothesis based on their relevance to the evidences ([27], [152] and [151]).

However, it is unrealistic and impossible to expect that “a priori” joint conditional probabilities between all the propositions in a BN are always available in the real world. The process of developing “a priori” knowledge is somehow “sophisticated” [95] and “arbitrary” [126]. Most Bayesian-based diagnostic models resort to reductionist approaches to relax the Bayesian restrictions. For example, it is often assumed that the fault variables are mutually exclusive and collectively exhaustive, and there is conditional independence of evidence given any hypothesis ([65], [49]).

### 2.4.3 Neural Network Models

Thagard developed a general algorithm for using an Artificial Neural Network (ANN) to perform abduction in his cognitive model ECHO ([125], [123]). In ECHO, hypotheses and evidences are represented by simple artificial neurons, which are connected by excitatory or inhibitory links corresponding to constraints between the propositions they represent. The general process of the algorithm is:

1. For every proposition, construct a neuron node representing the proposition in the network.

2. If a proposition supports another proposition, for example, a hypothesis explains an evidence, construct a symmetric excitatory link between the corresponding nodes.
3. If two propositions contradict each other, construct a symmetric inhibitory link between the corresponding nodes.
4. Assign each node an equal initial activation value, then update the activation value of all the nodes in parallel. The updated activation value of a node is calculated on the basis of its current activation value, the weights on links to other nodes, and the activation values of the nodes to which it is linked.
5. Continue the updating of activation value until all nodes have settled, i.e. achieved unchanging activation values.

After the above process has been completed, among all the nodes that represent potential hypotheses, the node which ends up with the highest positive activation value can be accepted as the best explanation of the evidence.

## 2.5 Summary

This chapter has reviewed existing works related to this thesis. The first part has introduced some significant ideas of explanation in Philosophy and Cognitive Science, which can be regarded as the theoretical foundations for building explanation models. The second part has dealt with existing works in building explanatory facilities for KBSs, which fall into two major categories according to the aspects of explanation models: generating basic content and providing adapted explanations. The

last section has reviewed existing models for “explaining available evidences”, which are usually regarded as independent KBSs that look for reasons for some observations (also called fact, evidence or symptom in different literatures), including three different types: rule-based model, probabilistic model and neural network model.

# Chapter 3

## Explanation Framework

### 3.1 Introduction

As previously discussed (Section 1.4.1), the objective of this thesis is to develop an computational framework for explanation generation in KBSs. An overview of the framework is outlined in this chapter as well as the introductions of the sub-models included in the framework.

The rest of this chapter is organized as follows: Section 3.2 explores some preliminary discussions relevant to the framework; Section 3.3 presents the general structure of the framework; Section 3.4 describes the development of EKB and relevant technologies; Section 3.5 highlights the reasoning issues regarding how to generate decision explanations in DEM; Section 3.6 outlines general ideas about how to perform software diagnostic in SDM; Section 3.7 summarizes the chapter.

## 3.2 Preliminaries

One of the major research objectives in this thesis is to provide explanations for all the three explanation focuses: “explaining available evidences”, “explaining models” and “explaining reasoning process”. We hereby give concrete definitions for these focuses particularly within our study:

- Decision explanation, which refers to “explaining reasoning process”, provides justification for how a KBS reaches its conclusions or how a KBS obtains the results. More specifically, to explain how a KBS reaches its conclusions (output of the KBS), based on the DN model (an explanation consists of a set of facts and a set of general laws), our system presents the facts (the input of the KBS, representing things that the user knows) together with some semantic rules, which deductively link the facts to the conclusions.
- Software diagnostic, which refers to “explaining available evidences”, presents reasons for the unexpected behaviors in software products, such as system failures, internal errors or warnings, and gives appropriate advice to solve the problems.
- Domain explanation, which refers to “explaining models”, provides detailed descriptive information of concepts and the relationships between the concepts within the domain (e.g. explanations for questions such as “what is that?” or “what does it mean?”).

Another research objective in our framework is to develop an independent explanation model that can reconstruct explanations for KBSs from a separate EKB. To

achieve this goal, several design principles are identified:

- Generating explanations is essentially a human-computer interaction. The knowledge representation method used in the EKB should have advances in information sharing between human and computers. Namely, EKB should have a shared and natural knowledge structure that captures human logics in an expressive and explicit way.
- To generate all the three types of explanations, EKB should: 1) represent all the domain concepts and relationships for decision explanations and software diagnostics; 2) capture logical rules that deductively link the input and output of KBSs for decision explanations; 3) encode detailed descriptions and definitions for domain concepts and relationships for domain explanations.
- The decision explanations and software diagnostics should be treated separately, as they have different focuses. Generally speaking, as DN model mentioned that an explanation of an event consists of facts and rules, decision explanations focus on uncovering the rules linking the facts and the event, while software diagnostics are mindful towards the facts that cause the event.

### **3.3 General Structure of the Framework**

Figure 3.1 shows the general structure of the framework. Generally, in this framework, both the KBS and the explanation model have knowledge bases respectively. The one in the KBS is used for making decisions and the EKB is developed for generating explanations. Information is shared between these two but in different formats.

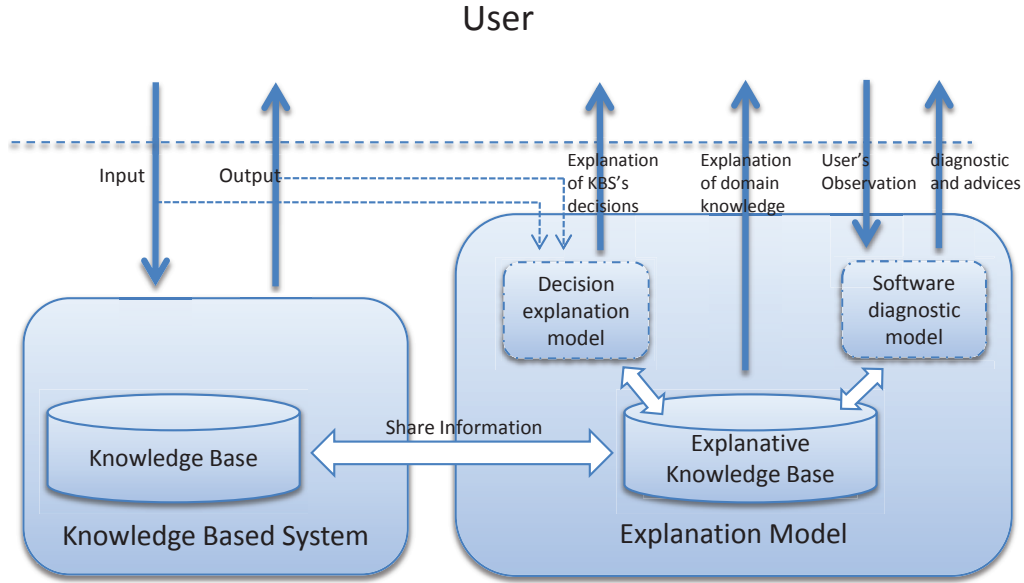


Figure 3.1: General structure of the explanation structure

The input and output of the KBS are together sent to DEM. DEM, then, backwards tracks the knowledge in EKB using a proper reasoner to logically connect the output and input of the KBS. An explanation about how KBS reaches its conclusions (output) from the input is consequently generated based on the knowledge track (namely the reasoning history). Once a user observes an unexpected system behavior, the software diagnostic model executes AR to generate explanations for what has been observed. An interaction process which handles context is considered. In addition, the information for domain explanations is statically encoded in EKB and can be directly provided to users upon request. In the following discussion, the three major sub-models, EKB, DEM and SDM, are introduced subsequently.

### 3.4 Explanative Knowledge Base (EKB)

Modern KBSs are weak in generating explanations, partly due to the fact that they may possess a very abstract knowledge base where human logics are not explicitly exist.

Considering the following example, [31] proposed a complex expert system for power transformer fault diagnosis, which comprises a Support Vector Machine (SVM) with Genetic Algorithm (GA) optimization. Such expert systems, which are combined with some advanced mathematic techniques (e.g. ANN [155] [141], BN [150] [157], Graph-based methods), are currently preferred by researchers in this area against early rule-based models [86] [77] for various considerations (e.g. authors of [150] believe that rule-based models have difficulties in determine the faulty section among multiple inferred possible choices and may lead to wrong conclusions). However, it has been argued that such systems lack explaining capability and behave as black boxes [20], since that explicit knowledge and human logic are not exist. In this case, although the early rule-based systems may not perform as good as the above advanced models, they proves there exists a chain of logic rules which links the observed symptoms and transformer faults. Such rule chains are usually regarded as explanations, and are usually easier to be located for explanation generation than for decision making (for explanation, the conclusion are given, so that the above mentioned problem of multiple choices and wrong conclusion is avoided). Therefore, constructing such a rule base only for explanation generation would be a solution for the black box problem, which is also the core idea of EKB.

To achieve this objective, the EKB in our framework should capture knowledge in



a more explicit and perspicuous way, so as to improve the information sharing between users and computers. For this reason, Semantic Web languages, OWL [82] and SWRL [58] are introduced. Semantic Web is motivated to represent web content in a form that is more easily machine-processable [7], so as to improve human-computer communications. Our EKB normally consists of an OWL ontology and several SWRL rules. The OWL ontology provides a shared, explicit and holistic view of domain knowledge and the SWRL rules are usually used to represent logics explicitly or implement some constraints. Information for domain explanation are also contained in the ontology as annotations of concepts or relationships.

This section mainly focuses on introducing OWL ontology and SWRL rules, including their definitions and syntax. An overview regarding how OWL and SWRL can be used to capture domain knowledge is also given, whereas the details will be discussed with a concrete application in Chapter 7.

### **3.4.1 OWL Ontology**

Ontology is essentially a philosophical study of the nature of existence. Problems, such as existence or classification of entities, are addressed in this study. In the context of knowledge representation, ontology refers to the specification of a conceptualization. It represents a domain in terms of concepts and the relationships between the concepts. An ontology usually holds a hierarchy, in which a concept may belong to a super-concept that is more abstract, and may also subsume several sub-concepts that are more concrete.

OWL has recently been recognized as the most popular language for constructing

an ontology [7]. OWL is a multi-disciplinary product, which integrates Philosophy, Knowledge Engineering and Computer Science. In general, OWL is a family of ontology description languages, which is based on Description Logic (DL) and is implemented using web technologies. The OWL family includes three sub-languages with different levels of expressiveness: OWL-Lite, OWL-DL and OWL-Full. OWL-DL is used in this study (OWL-DL is denoted by OWL in the following discussion).

An OWL ontology consists of three major components: class, individual and property. Classes represent types of objects in the domain. They are described using formal descriptions that precisely state the requirements for membership of a certain class. Classes are usually organized into a superclass-subclass hierarchy in OWL, which is also known as a taxonomy. Individuals refer to the actual objects in the domain. They can be instances of classes or fillers of properties. Properties are binary relations between individuals, namely they link two individuals together. A property also can represent a relationship between two classes if all the instances of both classes are linked by this property.

Figure 3.2 shows a simple example of ontology. “Person”, “Country” and “Pet” are classes, which have individuals as their instances. For example, the individuals “Gemma” and “Matthew” are instances of the class “Person”, while “Italy”, “England” and “USA” are instances of the class “Country”. The arrows linking individuals represent properties. For example, “livesInCountry” is a property that represents a binary relationship between “Person” and “Country”, so that the information “Matthew is a person who lives in country England” is captured in the ontology.

Beyond the three major components, OWL involves many constructs and ax-

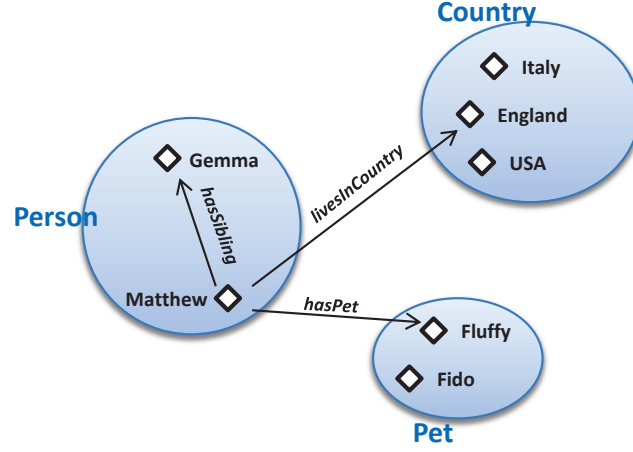


Figure 3.2: An example of OWL ontology

ioms for better expressiveness. The constructs and axioms are designed based on  $\mathcal{SHOIN}(\mathcal{D})$ , which is a subset of DL. The expressivity of  $\mathcal{SHOIN}(\mathcal{D})$  can be explained as follows:

- $\mathcal{S}$  is an abbreviation of  $\mathcal{ALC}$  with transitive roles.  $\mathcal{AL}$  represents the basic DL, which allows: 1) atomic concept negation; 2) concepts intersection; 3) universal restrictions; 4) limited existential quantification.  $\mathcal{C}$  represents complex concept negation.
- $\mathcal{H}$  refers to role hierarchy, which means super-property and sub-property are allowed.
- $\mathcal{O}$  means enumerated classes that are directly defined by a set of instances are allowed.
- $\mathcal{I}$  means description of inverse properties is supported.
- $\mathcal{N}$  means cardinality restrictions are supported.

Table 3.1: Syntax and semantics of OWL constructs and axioms

Constructor Name	OWL Syntax	DL Syntax	Semantics
atomic class	$A(URI)$	$A$	$A^I \subseteq \Delta^I$
object property	$R(URI)$	$R$	$R^I \subseteq \Delta^I \times \Delta^I$
datatype property	$U(URI)$	$U$	$U^I \subseteq \Delta^I \times \Delta_D^I$
individual	$o(URI)$	$o$	$o^I \in \Delta^I$
top	owl:Thing	$\top$	$\top^I = \Delta^I$
bottom	owl:Nothing	$\perp$	$\perp^I = \{ \}$
conjunction	intersectionOf( $C_1, C_2$ )	$C_1 \sqcap C_2$	$C_1^I \cap C_2^I$
disjunction	unionOf( $C_1, C_2$ )	$C_1 \sqcup C_2$	$C_1^I \cup C_2^I$
negation	complementOf( $C$ )	$\neg C$	$\Delta^I / C^I$
enumerated class	oneOf( $o_1, \dots$ )	$\{o_1, \dots\}$	$\{o_1^I, \dots\}$
existential quantification	restriction( $R$ someValuesFrom( $C$ ))	$\exists R. C$	$\{x   \exists y (x, y) \in R^I \cup y \in C^I\}$
universal quantification	restriction( $R$ allValuesFrom( $C$ ))	$\forall R. C$	$\{x   \forall y (x, y) \in R^I \rightarrow y \in C^I\}$
minimal cardinality	restriction( $R$ minCardinality ( $n$ ))	$\geq nR$	$\{x \in \Delta^I, \ y, (x, y) \in R^I\  \geq n\}$
maximal cardinality	restriction( $R$ maxCardinality ( $n$ ))	$\leq nR$	$\{x \in \Delta^I, \ y, (x, y) \in R^I\  \leq n\}$
Axiom Name	OWL Syntax	DL Syntax	Semantics
concept inclusion	subClassOf( $C_1, C_2$ )	$C_1 \sqsubseteq C_2$	$C_1^I \subseteq C_2^I$
equivalent classes	equivalentClasses( $C_1, C_2$ )	$C_1 \equiv C_2$	$C_1^I = C_2^I$
disjoint classes	disjointClasses( $C_1, C_2$ )	$C_1 \sqcap C_2 = \perp$	$C_1^I \cap C_2^I = \emptyset$
property inclusion	subPropertyOf( $R_1, R_2$ )	$R_1 \sqsubseteq R_2$	$R_1^I \subseteq R_2^I$
property transitivity	[Transitive]	$Tr(R)$	$R^I = (R^I)^+$
inverse properties	[inverseOf( $R_0$ )]	$R \equiv (R_0^-)$	$R^I = (R_0^I)^-$

- ( $\mathcal{D}$ ) means datatype properties, data values or data types can be used.

Syntax and semantics of major constructs and axioms of OWL are listed in Table 3.1.

OWL is also regarded as an extension of Resource Description Framework (RDF). RDF is a XML-based language that focuses on describing web resources. The general idea of RDF is to represent object-attribute-value statement. OWL is developed on top of RDF with higher expressive power and stronger reasoning support. In fact, all the OWL elements are represented by RDF resources in the RDF level. RDF offers

a set of utility properties to annotate RDF resources, giving the potential to encode descriptive information for the purpose of providing domain explanations:

- `rdfs:seeAlso`: relates a resource to another resource that explains it.
- `rdfs:isDefinedBy`: relates a resource to the place where its definition is found.
- `rdfs:comment`: is used to provide additional information that is associated with a resource for human readers.
- `rdfs:label`: is used to provide a human-friendly label that is associated with a resource.

### 3.4.2 SWRL Rules

OWL is an ideal ontology description language which has considerable expressive power, especially in representing concepts and their relationships between each other. However, OWL also has expressive limitations. The language includes a relatively rich set of class constructs, but the methods provided for talking about properties is much weaker [57]. One particular weakness is the incapability of handling property composition. For example, it is difficult to describe the relationship between the composition of the “parent” and “brother” properties and the “uncle” property. One way to address this problem, that has already been proposed in OWL2, is to introduce a new axiom called property chain. However the property chain can only capture simple property composition. Complex relationships, including, for example, cardinality restrictions, are hardly represented by the property chain.

Table 3.2: Syntax and semantics of SWRL atoms

Atom Name	SWRL Syntax	SWRL Semantics
Class	$C(x)$	$S(x) \in EC(C)$
Object property	$P(x, y)$	$\langle S(x), S(y) \rangle \in ER(P)$
Datatype Property	$Q(x, z)$	$\langle S(x), L(z) \rangle \in ER(Q)$
Same objects	$sameAs(x, y)$	$S(x) = S(y)$
Different objects	$differentFrom(x, y)$	$S(x) \neq S(y)$
builtin functions	$builtIn(r, z1, \dots, zn)$	$\langle S(z1), \dots, S(zn) \rangle \in D(f)$

An alternative way to overcome the above expressive limitations is to extend OWL with some form of “rule language”. SWRL is thus proposed [57]. SWRL is generally a combination of OWL with the Unary/Binary Datalog RuleML language [58]. The language allows users to write horn-like rules that can be expressed in terms of OWL constructs. A rule takes the form of an implication between an antecedent (body) and a consequent (head). A rule body may contain multiple atoms, and is treated as a conjunction of these atoms. The intended meaning of a rule can be read as: whenever the atoms specified in the antecedent hold, the atom in the consequent must also hold.

The syntax and semantics of atoms in SWRL are listed in Table 3.2, where  $EC$  is a mapping from classes and datatypes to a set of ontology resources and a set of literal values respectively,  $ER$  is a mapping from properties to a set of binary relations,  $L$  is the mapping from typed literals to elements in a set of literal values, and  $S$  is a mapping from individual names to the actual individuals in the ontology.

### 3.4.3 Capturing Domain Knowledge

With OWL and SWRL, we hereby describe a general method for representing domain knowledge in EKB. Details of the mapping will be discussed with a concrete

application in Chapter 7.

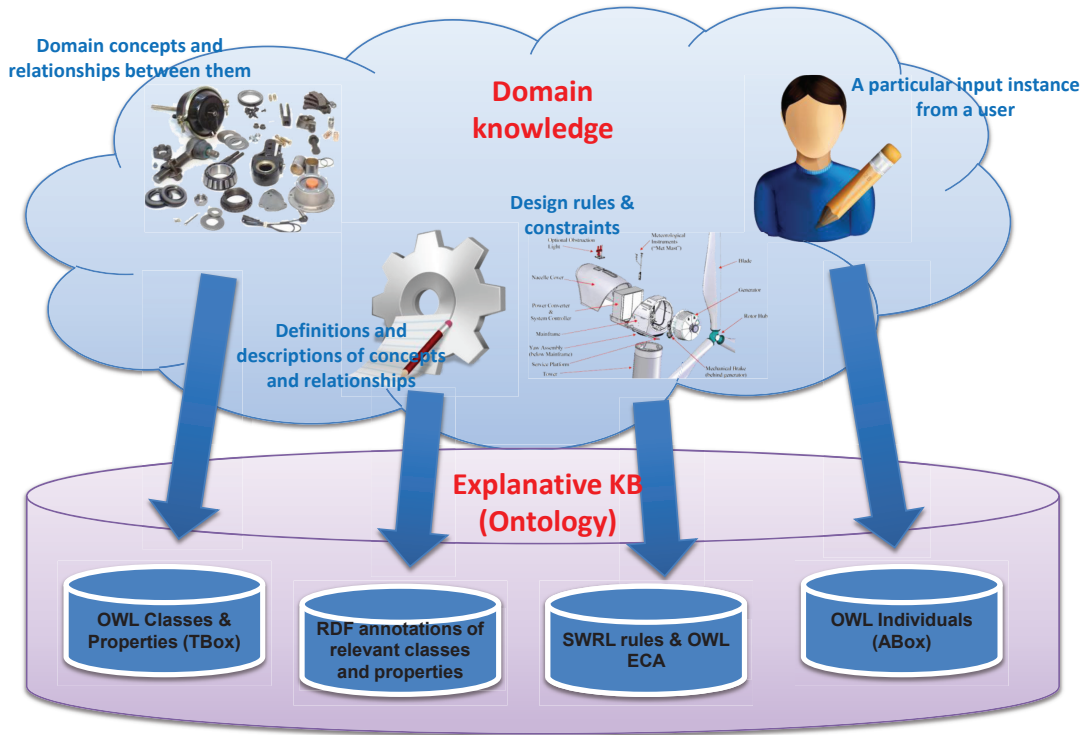


Figure 3.3: Capturing domain knowledge using OWL and SWRL

As shown in Figure 3.3, the domain knowledge can be divided into four parts:

1. Domain concepts and relationships, which refer to the information of the conceptualization of the domain, such as concept identities, subsumption hierarchies of concepts, relationships between concepts, and properties of the relationships (whether they are transitive, functional or symmetric). Such information is also called TBox, which are captured using OWL constructs regarding classes and properties, representing the terminological knowledge within the domain.
2. Definitions and descriptions of the concepts and relationships, which refer to the

descriptive information, such as detailed definitions or references for a particular concept or relationship. Such information is encoded using RDF utility properties (see Section 3.4.1) as annotations of relevant concepts or relationships, and can be directly accessed once a user asks for domain explanations.

3. Design rules and constraints, which refer to a human logic based presentation about how a KBS achieved its results (output) from the input facts, such as how the system moves from the current state to the next state and what are the rules underpinning such movements. These logics are represented using SWRL rules and OWL Equivalent Classes Axiom (ECA) which implicitly represents a rule by claiming that two class expressions are equivalent.
4. Input instances from the user, which refer to the facts that the user already knows in a particular case, can also be seen as an instantiation of the domain concepts and relationships. Such information is transferred to the ABox of the ontology, representing assertion knowledge within the domain.

### **3.5 Decision Explanation Model (DEM)**

DEM reasons through EKB to logically connect the output and the input of a KBS, and provides the resulting logical sequence as the explanation to how the KBS obtains the output. Seeing as human logics are explicitly captured using rules in our model, the logical connection here means a rule trace which links the input facts and output results. Thus, how to build a reasoner that can reason through OWL and SWRL to locate the rule trace is the key in DEM. In the following discussion, a quick review of



existing OWL reasoners are firstly given, followed by a proposal of a novel reasoner that is used for explanation generation.

### 3.5.1 Existing OWL Reasoners

Many reasoners have been developed for OWL ontology and some of them support SWRL rules. Existing reasoners can be generally classified into two categories according to their reasoning algorithms: rule-based algorithm and tableau-based algorithm.

Existing rule-based reasoners, include Bossam [60], Jena[53], SweetRules[94] and KAON2[59]. They tend to translate OWL clauses into rules, and then apply optimized rule-based algorithms for reasoning tasks. Among them, Bossam translates OWL axioms into a set of OWL inference rules using a self-defined Bossam rule language with extended expressiveness. The classical RETE algorithm is applied on the Bossam rule base for major reasoning tasks. Jena is actually a JAVA framework for building Semantic Web applications, and supports OWL inference by translating OWL into RDFS rules. SweetRules engine is an indirect inferencing engine which only focuses on translating OWL to other rule languages such as Jess or Prolog, and uses their reasoning engine to finish the jobs. In KAON2, OWL ontology can be reduced to a disjunctive datalog program and well known deductive database techniques, such as magic sets or join-order optimization, are used to improve the performance.

The other category of reasoners, including Pellet [117], RacerPro [41] and FaCT++ [133], implements conventional DL tableau algorithm. Tableau algorithm basically reduces all the reasoning tasks to the satisfiability check of a concept expression. For example, a subsumption can be reduced to satisfiability as:

$$C \sqsubseteq D \leftrightarrow C \sqcap \neg D \text{ is not satisfiable}$$

After a concept satisfiability problem is constructed, the algorithm attempts to find an interpretation that satisfies the concept. The interpretation is incrementally constructed as a “tableau”. The general process is listed as follows:

- Transform the concept expression in to Negation Normal Form (NNF), namely, negation occurs only in front of concept names. The NNF transferring rules are listed below:

$$- \neg(C_1 \sqcup C_2) \equiv \neg C_1 \sqcap \neg C_2$$

$$- \neg(C_1 \sqcap C_2) \equiv \neg C_1 \sqcup \neg C_2$$

$$- \neg \exists R.C \equiv \forall R.\neg C$$

$$- \neg \forall R.C \equiv \exists R.\neg C$$

$$- \neg \neg C \equiv C$$

- The algorithm initializes the transformed expression  $C$  with a single assertion,  $A = C(x_0)$  ( $A$  denotes the initial ABox). The concept expression is then expanded using a set of consistency-preserving transformation rules defined below, until either a clash is found, or the rules can no longer be applied.

$$- \text{AND-rule1: } \mathbf{IF} \ A \text{ contains } (C_1 \sqcap C_2)(x), \text{ but does not contain } C_1(x), \\ \mathbf{THEN} \ A \rightarrow A \cup C_1(x)$$

$$- \text{AND-rule2: } \mathbf{IF} \ A \text{ contains } (C_1 \sqcap C_2)(x), \text{ but does not contain } C_2(x), \\ \mathbf{THEN} \ A \rightarrow A \cup C_2(x)$$

- OR-rule: **IF**  $A$  contains  $(C_1 \sqcup C_2)(x)$ , but contains neither  $C_1(x)$  nor  $C_2(x)$ ,  
**THEN**  $A \rightarrow A \cup C_1(x)$  and  $A \rightarrow A \cup C_2(x)$ ,
  - EXISTS-rule: **IF**  $A$  contains  $\exists R.B(x)$ , and there is no  $y$  such that  $R(x, y) \in A$  and  $B(y) \in A$ , **THEN**  $A \rightarrow A \cup \{R(x, y), B(y)\}$  for a new  $y$
  - FORALL-rule: **IF**  $A$  contains  $\forall R.B(x)$ , and there is a  $y$  such that  $R(x, y) \in A$  and  $B(y) \notin A$ , **THEN**  $A \rightarrow A \cup \{B(y)\}$
- The algorithm terminates once either a clash is found, or the rules can no longer be applied. An expanded ABox is said to contain a clash, if and only if, for a concept  $C$  and an instance  $x$ ,  $C(x)$  and  $\neg C(x)$  are both included. Otherwise it is clash-free. A concept expression  $C$  is satisfiable if and only if there exists an interpretation in  $\mathcal{A}$  that is clash-free.

### 3.5.2 Developing a Novel Reasoner

Existing reasoners that were mentioned above are all developed for general reasoning purposes for Semantic Web. Thus, they are not efficient in generating explanations. We hereby propose a novel ontology ABox reasoner, BCAR.

BCAR is an ABox query reasoner, originally designed to retrieve instances which implicitly belong to some defined concepts. Defined concepts, here, refer to concepts which are defined by rules. Generally speaking, to retrieve instances for these defined concepts, BCAR automatically searches for solutions for the conditions of the relevant rules (a solution is a combination of the facts which makes all the condition atoms hold true) in the ABox, and generates query results based on the solutions that have been found.

The usage of BCAR in generating explanations is similar. To explain why a rule consequent holds true, BCAR first searches the ABox to determine whether there is a solution for the rule conditions. If there is a solution, the facts contained in the solution are then presented to the user together with the rule as the explanation.

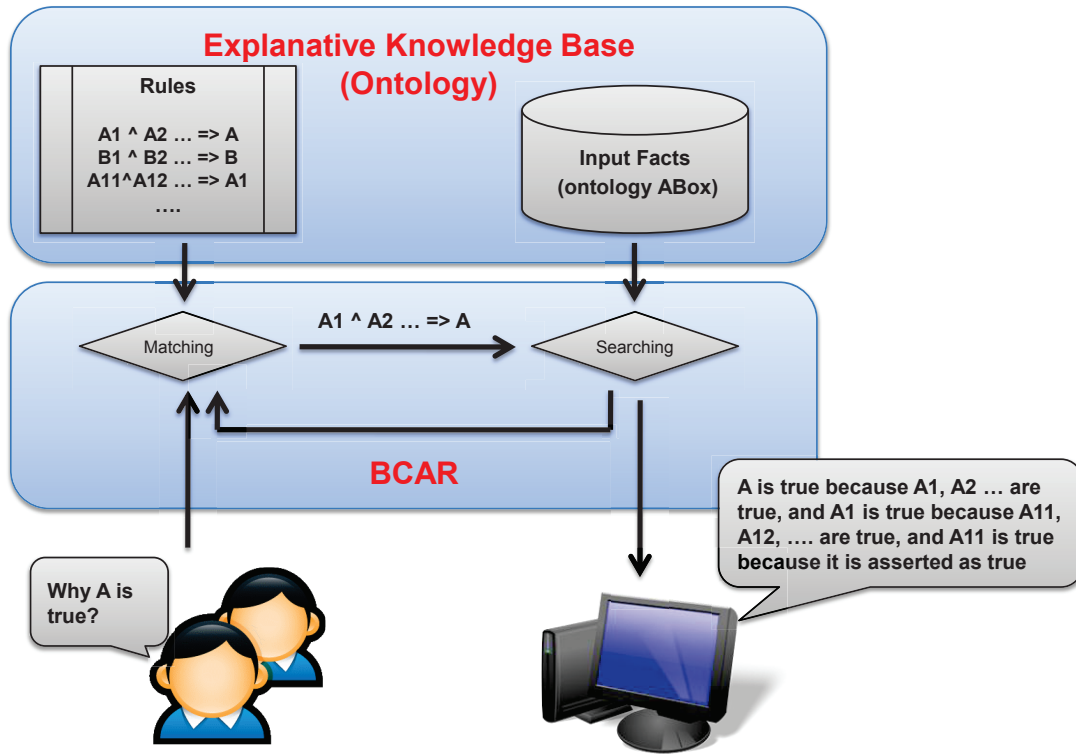


Figure 3.4: BCAR in generating decision explanations

Figure 3.4 illustrates a simple example showing the role of BCAR in generating explanations. In the example, a KBS has made a decision saying that “A is true”. A user then needs the explanation model to explain “why A is true?”. The reasoner first fires the rule whose consequent matches the goal, as shown in the figure. Then it needs to search the ABox in order to find a solution which proves that all the condition

atoms “A1”, “A2”, ..., are true. Some of the atoms may be directly asserted as true, while the other atoms, e.g. “A1”, act as sub-goals needed to be explained, to which it then activates some other rules, e.g. “ $A_{11} \wedge A_{12} \dots \rightarrow A_1$ ”. BCAR keeps repeating this process until all the atoms are directly explained by the facts, and, lastly, it can output the explanation, as shown in the figure.

Comparing with other existing reasoners, BCAR has following features:

1. BCAR interprets OWL and SWRL under Closed World Assumption and Unique Name Assumption (only for ABox) to improve the reasoning performance for closed world based information models.
2. Some existing OWL reasoners support SWRL by translating SWRL to some other formats (Jess, Jena or Sesame) ([84], [94]). BCAR directly works on SWRL without any translation, which improves its efficiency.
3. BCAR adopts backward reasoning, similar to the Prolog derivation tree. The reordering technique from Optimized Conjunctive Query (OCQ) [116] is integrated to improve the performance

Details of BCAR are discussed in Chapter 4.

## 3.6 Software Diagnostic Model

The software diagnostic model diagnoses user observations of unexpected behaviors within a system. The concept of diagnostic is based on AR. AR is a form of inference that generates a hypothesis that best explains an observation [140]. Traditionally, AR in most of the existing diagnostic systems is only a one time process. However,

the process of diagnosis is actually a multiple run interactive process in real life. Considering the example of a patient seeing a doctor, the doctor firstly generates a hypothesis based on the patient's initial description of his or her symptoms. He or she then needs to ask the patient for further information to confirm the hypothesis. If the hypothesis is supported by the new information, the doctor then has a stronger belief of the hypothesis; otherwise, he or she may generate another hypothesis based on the new information. This interaction will keep going until the doctor is largely convinced that a particular hypothesis is the correct one. In the following discussion, a brief introduction of AR is firstly given, followed by a general discussion about how to implement interaction into AR. The details of the implementation are described in Chapter 5.

### 3.6.1 Abductive Reasoning

Abduction (or inference to the best explanation) is a form of inference that generates a hypothesis best explaining the observation [63]. The process of AR involves identifying the symptoms (also called observations or evidences) which needs to be explained, generating hypotheses that potentially explain the evidence, evaluating competing hypotheses and selecting the best explanation according to some criteria. Pople [100] defined the AR within the following schema:

- rule:  $\forall[P(x) \rightarrow Q(x)]$
- case:  $P(a)$
- conclusion:  $Q(a)$

Deductive Reasoning (DR) is from the rule and the case to the conclusion, namely if both the rule and the case  $P(a)$  is true,  $Q(a)$  is then concluded. Inductive Reasoning (IR) is from the case and the conclusion to the rule. If  $P(a)$  and  $Q(a)$  are both observed to be true, we may hypothesize that “perhaps all things  $P$  are also  $Q$ ”. AR refers to the third option - it direct from the fact and the rule to the case. If  $Q(a)$  is observed to be true, and the rule “all things  $P$  are  $Q$ ” is known to be true, “perhaps  $a$  is  $P$ ” is then hypothesized.

There are four computational paradigms to approach AR [128]: 1) the deductive paradigm implemented in logic or rule-based systems; 2) the schematic approach using explanation patterns or analogies; 3) the probabilistic method implemented using BN; and 4) the connectionist approach built from ANN. Existing models for these paradigms have been reviewed in Section 2.4. Among these models, the probabilistic approaches provide a quantitative base allowing us to model human diagnostic processes under uncertain circumstances. As is discussed in Section 2.4.2, a fundamental research challenge for implementing the Bayesian theory is to handle missing joint probabilistic information, so that most Bayesian-based systems resort to approaches that relax the Bayesian assumptions [95] [126].

### 3.6.2 Involving Interaction

Interactions are defined as mutual influences between the two coupled dynamic systems. For example, an agent and its environment are jointly responsible for the agent’s behavior [12]. The agent interacts with the environment so as to adaptively fit the constraints of the trajectory of one large coupled dynamic system [13]. With-

out interaction, algorithms would produce direct mappings completely determined by their inputs, which are memoryless and history-independent. In contrast, interactive systems that provide history dependent services can, over time, learn from and adapt to experience [142]. Interactive approaches provide supplementary means to the algorithmic computation paradigm. The implementation of the interactive approaches can be traced back to diagnostic systems built 30 years ago. For example, the MEDAS system [11], which was designed to provide clinicians with decision supports, can identifies medical disorders, and reasons through the associated unknown features, and, subsequently, interacts with users to locate clinical relevant features.

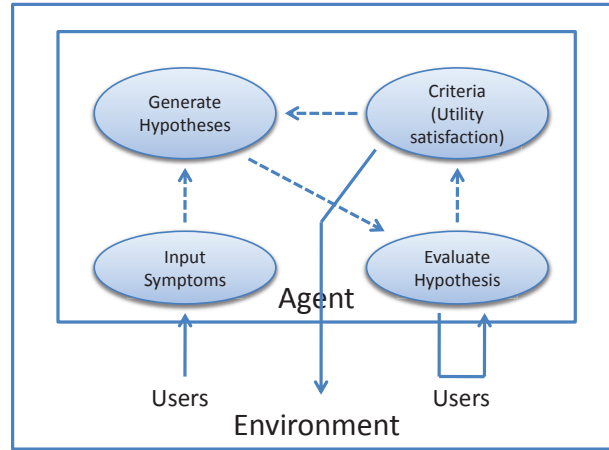


Figure 3.5: The general process of interactive AR

The process of AR with interaction is illustrated in Figure 3.5. Users provide feedback and new information, through which an agent can re-generate hypotheses that maximize its knowledge fitness in respect to uncertainty reduction.

MRICN is designated to provide this interactive AR. This certainty network is developed based on Bayesian theory and Opinion Pooling (OP) approach. To differ-



entiate the Bayesian approaches in which probability is used to represent the degree of a system's trust of a hypothesis, MRICN adopts the notion of "certainty" to represent this degree of trust. When the generated hypothesis is not valid within the interaction, a search algorithm is involved to look for new symptoms that provide the maximum certainty gain. The system subsequently interacts with the environment to acquire feedback for this new symptom. Details of MRICN will be discussed in Chapter 5.

### 3.7 Summary

In this chapter, an innovative and novel explanation framework has been introduced. The framework consists of three major components: EKB, DEM and SDM. In EKB, an OWL/SWRL ontology is built to capture the domain knowledge. The knowledge for providing domain explanations is encoded statically in the ontology as RDF annotations. DEM reconstructs decision explanations by reasoning through EKB to logically connect the inputs of the KBS and the decision that has been made. Existing ontology reasoners have also been reviewed. A novel reasoner, BCAR, has been proposed. SDM interactively diagnoses user observations of a system's unexpected behaviors to explain mismatches between system behaviors and user expectations, which is one of the major reasons why people need explanations. The diagnostic is based on AR. General ideas of AR have been discussed. The importance of involving interactions into traditional AR has been identified.

Traditionally, most of the explanation models are viewed as sub-models or functional components of the main KBSs. Different with such models which are quite

application-oriented and heavily constrained by the main system, the proposed framework possesses a complete new structure that allows the explanation model to stand outside the main system and is possible to adapt to many applications. This general applicability is one of the major merits of the proposed framework. In addition, the introduction of ontology techniques brings better quality explanations than traditional rule-traces since that ontology provides a holistic view of knowledge. Thirdly, as previously mentioned, the explanation model in the framework firstly covers all the three explanation focuses: “explaining models”, “explaining reasoning process” and “explaining available evidences”.

This chapter is just an overview of the framework. General ideas regarding the structure of the framework and how the sub-models work have been given, as well as some background knowledge of relevant technologies. Detailed methods and algorithms, including BCAR and MRICN, will be discussed in the following chapters.

# Chapter 4

## Backward Chained ABox Reasoner

### 4.1 Introduction

As is discussed in Chapter 3, DEM uses a proper reasoner to logically connect the output and input of a KBS. Explanations about how the KBS reaches conclusions (output) from the input are consequently generated based on the knowledge track (namely the reasoning history) in EKB. A novel reasoner, BCAR, has been proposed to carry out this task. Features of BCAR have been highlighted. This chapter mainly focuses on the technical details of BCAR.

The rest of this chapter is organized as follows: Section 4.2 sets out the preliminary discussion; Section 4.3 outlines the concept framework of BCAR; Section 4.4 explores how to build the unified rule base; Section 4.5 presents the core reasoning algorithm; Section 4.6 describes how to handle some special rules or atoms in the reasoner; Section 4.7 tests BCAR with a benchmark ontology, and shows the experiment result; Section 4.8 summarizes the chapter.

## 4.2 Preliminaries

In general, BCAR is an ABox query reasoner for OWL ontology with SWRL rules. The objective of BCAR is to help users retrieve implicit instances or fillers for some defined classes or properties. Different from existing OWL reasoners, BCAR interprets OWL and SWRL under Closed World Assumption (CWA) and Unique Name Assumption (UNA). The inspiration for our research comes from the fact that OWL ontology has been widely used to capture closed world based information models ([71], [5], [149], [36] and [156]), despite all existing reasoners holding Open World Assumption (OWA). Because of this, applying an OWA-based reasoner to CWA-based information model will produce some reasoning errors. As such, BCAR is proposed to address the problem.

More specifically, BCAR has to hold the following assumptions:

**Assumption 1:** The ontology contains only the explicitly expressed individuals. There is not any unknown individual.

**Assumption 2:** Only defined classes (or properties) are allowed to have implicit instances (or fillers). Other classes (or properties) only contain instances (or fillers) that are explicitly expressed.

**Assumption 3:** If a class (or property) cannot be proved to contain an instance (or filler), then the class (or property) does not contain the instance (or filler). If a class (or property) cannot be proved to contain any instance (or filler), then the class (or property) contains nothing (negation as failure).

**Assumption 4:** Two individuals are the same if and only if their names are

the same.

In OWL/SWRL ontologies, the most two popular methods to define classes and properties are SWRL rules and OWL ECA. BCAR only considers classes (or properties) which have corresponding ECAs or appear in the heads of SWRL rules as defined classes (or properties), and the corresponding ECAs and SWRL rules are their definitions.

### 4.3 Concept Framework

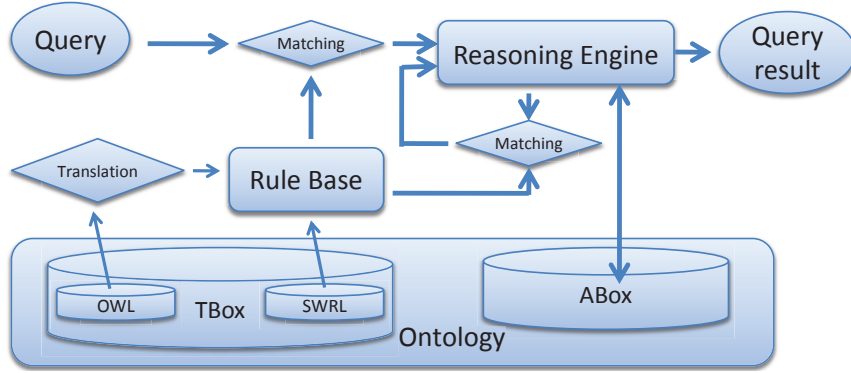


Figure 4.1: Concept framework of BCAR

Figure 4.1 shows the concept framework. BCAR firstly creates a unified rule base with SWRL rules and OWL ECAs. Once a query has been inputted, the rule which matches the query fires. The reasoning engine is then applied to the fired rule to search for its solutions. The engine requires information from the ABox and also may fire some other rules if required. After the searching is finished, the query result is, then, generated based on the obtained solutions.

## 4.4 Building Rule Base

As mentioned above, BCAR only accepts two ways to define classes and properties: SWRL rule and ECA. Because the backward chained reasoning engine requires a unified rule base, there is a need to translate ECAs to SWRL-like rules, so that the process of building the rule base for BCAR is actually the process of translating all the OWL ECAs to SWRL rules.

Originally ECA refers to a type of axiom representing bidirectional equivalency relationship between two classes, and is usually used to describe the necessary and sufficient condition for an atomic class, indicating that the class is equivalent to another complex class expression. In the case of rule representation, an ECA can be read as: if an instance holds all the conditions specified in the complex class expression, it then belongs to the atomic class, and vice versa.

Normally, ECAs cannot be translated to rules directly, as they represent bidirectional relationships. However, BCAR considers ECAs only as class definition rules, and processes ECAs equally with SWRL rules. In this case, interpreting ECA as one direction logic, from the definition to the class, is reasonable.

The translation is generally based on the First Order Logic (FOL) semantics of the OWL constructs. Considering the following ECA (written in DL syntax) as an example:

$$\exists R.B \equiv C$$

According to [8], the left side of this ECA can be translated to the following FOL formula ( $\phi$  maps OWL class  $B$  and property  $R$  into unary and binary predicates  $\phi C(x)$  and  $\phi R(x, y)$  respectively):

Table 4.1: Translating ECA to SWRL-like rules

Construct Name	ECA Syntax	Rules
owl: intersectionOf	$C = A \text{ and } B$	$A(?x) \wedge B(?x) \rightarrow C(?x)$
owl: unionOf	$C = A \text{ or } B$	$A(?x) \rightarrow C(?x)$ $B(?x) \rightarrow C(?x)$
owl: complementOf	$C = \text{not } A$	$\text{Not } (A(?x)) \rightarrow C(?x)$
owl: someValuesFrom	$C = R \text{ some } A$	$R(?x, ?y) \wedge A(?y) \rightarrow C(?x)$
owl: allValuesFrom	$C = R \text{ only } A$	$\forall ?y (R(?x, ?y) \wedge A(?y)) \rightarrow C(?x)$
owl: hasValue	$C = R \text{ has } I$	$R(?x, I) \rightarrow C(?x)$
owl: minCardinality	$C = R \text{ min } n$	$(R \geq n) (?x) \rightarrow C(?x)$
owl: maxCardinality	$C = R \text{ max } n$	$(R \leq n) (?x) \rightarrow C(?x)$
owl: cardinality	$C = R \text{ exactly } n$	$(R = n) (?x) \rightarrow C(?x)$
Complex ECA	$C = A \text{ and } (R \text{ some } B)$	$R(?x, ?y) \wedge B(?y) \rightarrow H(?x)$ (create an intermediate class H) $A(?x) \wedge H(?x) \rightarrow C(?x)$

$$\exists x(\phi R(x, y) \wedge \phi B(y))$$

Since that BCAR regards the bi-directional equivalence as one-directional implication, the whole ECA can be transferred to:

$$\forall x(\phi R(x, y) \wedge \phi B(y) \rightarrow \phi C(x))$$

which is exactly the FOL semantic of the SWRL rule:

$$R(?x, ?y) \wedge B(?y) \rightarrow C(?x)$$

The translation is then completed. Other translations are described in Table 4.1.

(A, B, C are classes, P is a property, I is an individual and  $n$  is a number)

## 4.5 Reasoning Algorithm

In BCAR, retrieving instances (or fillers) of defined classes (or properties) activates their definition rules. The process of query is then transformed into a process of

searching the ABox for solutions for the definition rules. Reasoning results are generated based on the solutions. In the following discussion, the solution is firstly defined, the searching algorithm is then given, followed by a description of how to generate results based on solutions, and the backward chaining in the rule base is finally discussed.

For the purpose of simplicity, we introduce the following terms to represent reasoning tasks ( $\mathcal{C}$  and  $\mathcal{P}$  denote defined class and property respectively):

$\mathcal{C}(?)$  : query for all the instances of  $\mathcal{C}$

$\mathcal{C}(? = a)$  : check whether  $a$  is an instance of  $\mathcal{C}$

$\mathcal{P}(?, ?)$  : query for all the fillers of property  $\mathcal{P}$

$\mathcal{P}(? = a, ?)$  : query for the instances which are the property values of  $a$  for  $\mathcal{P}$

$\mathcal{P}(?, ? = a)$  : query for the instances whose property value is  $a$  for  $\mathcal{P}$

$\mathcal{P}(? = a, ? = b)$  : check whether  $(a, b)$  is a filler of property  $\mathcal{P}$

The example ontology given below is for the following algorithm description:

#### **TBox**

Atomic Classes:  $A; B; C; D$

Defined Class:  $E = OP4 \text{ some } C$ ;

Object Properties:  $OP1(\text{Domain} : A, \text{Range} : B); OP2(\text{Domain} : B, \text{Range} : C, D)$

Defined Object Property:  $OP4(\text{Domain} : A, \text{Range} : C)$

rule 1(SWRL):  $OP1(?x, ?y) \wedge OP2(?y, ?z) \wedge C(?z) \rightarrow OP4(?x, ?z)$

#### **ABox**

$A\{A1; A2; A3\}; B\{B1; B2; B3; B4\}; C\{C1; C2; C3; C4\}; D\{D1; D2\}$

$OP1\{(A1, B1); (A1, B2); (A2, B3); (A3, B3); (A3, B4)\}$

$OP2\{(B2, C2); (B2, C3); (B3, D1); (B4, C4)\}$



### 4.5.1 Solution

In a solution, each variable in the rule is bound to a value (value can be individual or datatype value such as integer and string), which makes all the atoms in the rule body hold true. In the example ontology, rule 1 has three variables:  $?x, ?y, ?z$ . With the above ABox, solutions for rule 1 are:

$$\{?x \leftarrow A1; ?y \leftarrow B2; ?z \leftarrow C2\}$$

$$\{?x \leftarrow A1; ?y \leftarrow B2; ?z \leftarrow C3\}$$

$$\{?x \leftarrow A3; ?y \leftarrow B4; ?z \leftarrow C4\}$$

All the reasoning results of BCAR is generated based on solutions, details of which are discussed later.

### 4.5.2 Searching for Solutions

Searching for solutions is the key of BCAR. The searching process in BCAR is generally an OCQ process based on the well-known Prolog derivation tree. The idea of atoms reordering [116] is integrated to improve the performance.

Taking the example ontology, assuming the reasoning task is  $OP4(?, ?)$ , rule 1 fires. The search is then started with the following steps:

1. Preprocessing: Build a Temporary Atom List (TAL) from the rule body. Define a Value Range (VR) for each variable based on the Assumption 2. In the example, from rule 1, it is easy to figure out that  $?x$  belongs to  $A$ ,  $?y$  belongs to  $B$ , and  $?z$  belongs to  $C$ . Based on Assumption 2,  $?x$ ,  $?y$  and  $?z$  can only be the explicitly asserted members of  $A$ ,  $B$  and  $C$  respectively. Figure 4.2 shows the TAL and VRs of the example. Other cases are listed below:

- (a) The VR of  $?x$  (or  $?y$ ) contains only  $A1$  (or  $B1$ ) if the reasoning task is  $OP4(? = A1, ?)$  (or  $OP4(?, ? = B1)$ );
- (b) For a constant, BCAR creates a new variable whose VR contains only the constant's value;
- (c) For variable belonging to a defined class, the  $VR = \top$  (Assumption3);
- (d) For datatype variable, the  $VR = \infty$

$OP1 \wedge OP2 \wedge C$			Temporary Atom List
$?x <-$	$?y <-$	$?z <-$	Value binding
$A1, A2,$ $A3$	$B1, B2,$ $B3, B4$	$C1, C2,$ $C3, C4$	Value range

Figure 4.2: The root of searching tree

2. Variable choosing and branching: The performance of CQ relies heavily on the query ordering. In BCAR, the reasoning always starts from the variable which has the minimal size of VR and has not been bound to a value, so that the number of branches of the searching tree is minimized. The selected variable is called SV. In the example,  $?x$  is SV, since it has the minimal size of VR among all the variables. BCAR then generates branches for each value in VR of  $?x$ .
3. Binding and intersecting: In each branch, the SV is bound to a value. BCAR then processes the atoms related to the SV in TAL, which may reduce the size of VR of SV or other variables related to SV by an intersecting process. After that, the processed atoms are removed from TAL. In the example,  $?x$  is bound to  $A1$  in the first branch. BCAR then processes  $OP1(?x(= A1), ?y)$  which is

the only atom related to  $?x$  in TAL. Based on the ABox and Assumption 3,  $?y$  can only be either  $B1$  or  $B2$ . BCAR then intersects  $\{B1, B2\}$  with  $?y$ 's original VR  $\{B1, B2, B3, B4\}$  to be the new VR of  $?y$ , as Figure 4.3 shows:

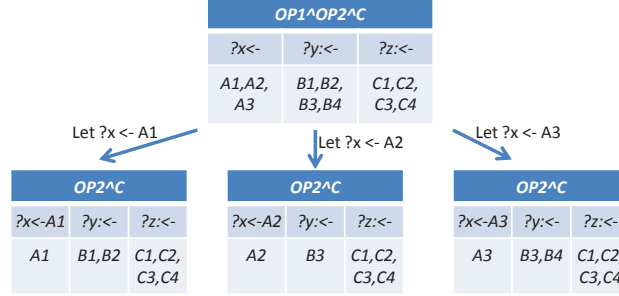


Figure 4.3: Generating branches

4. Termination: BCAR repeats the above step 2 and step 3 until: 1)VR of any variable turns out to be empty (based on Assumption 1&2, it means this variable is unsolvable in this branch); 2)Every variable has been bound to a value (a solution has been found in this branch); Figure 4.4. shows how the search tree finds solutions and how it is terminated. Since that the size of VR of a variable is bounded by the size of ontology ABox, the algorithm is guaranteed to terminate.

### 4.5.3 Generating Reasoning Results

BCAR generates results for every reasoning task as follows (assuming the head of the definition rule of  $\mathcal{C}$  and  $\mathcal{P}$  are  $\mathcal{C}(?x)$  and  $\mathcal{P}(?x, ?y)$  respectively):

1.  $\mathcal{C}(?)$  : returns all the values of  $?x$  from all the solutions;

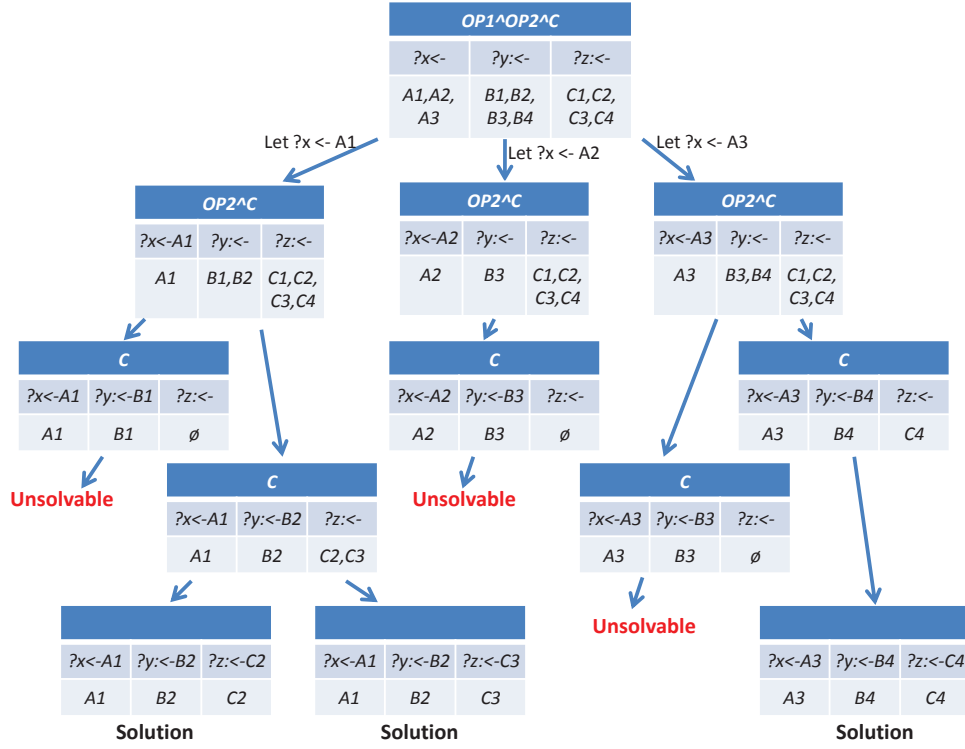


Figure 4.4: Searching tree in BCAR

2.  $\mathcal{C}(? = a)$  : sets the initial VR of  $?x$  to be  $\{a\}$ , checks whether a solution can be found;
3.  $\mathcal{P}(?, ?)$  : picks up all the values of  $?x$  and  $?y$  from all the solutions;
4.  $\mathcal{P}(? = a, ?)$  (or  $\mathcal{P}(?, a)$ ) : sets the initial VR of  $?x$  (or  $?y$ ) to be  $\{a\}$ , returns all the values of  $?y$  (or  $?x$ ) from all the solutions;
5.  $\mathcal{P}(? = a, ? = b)$  : sets the initial VR of  $?x$  and  $?y$  to be  $\{a\}$  and  $\{b\}$  respectively, checks whether a solution can be found;

#### 4.5.4 Backward Chaining

In the “Binding and intersecting” step of the searching process, if BCAR needs to process an atom corresponding to a defined class or property, another rule fires. For example, considering the defined class  $E = OP4 \text{ some } A$ , from Table 4.1 the translation rule is  $OP4(?x, ?y) \wedge D(?y) \rightarrow E(?x)$ . Assuming the reasoning task (goal) is  $E(? = A1)$ , while searching for the solutions for the translation rule, the “Binding and intersecting” step will add another reasoning task (new goal)  $OP4(? = A1, ?)$  and rule 1 consequently fires. This is the so-called backward chained reasoning.

## 4.6 Handling Special Rules and Atoms

In Table 4.1, some of the translation rules can be solved normally using the above algorithm (e.g. owl:intersectionOf; owl:unionOf; owl:someValuesFrom), while the others can not. This section generally discusses how BCAR handles some special rules and atoms.

- owl:complementOf

$C = \text{not}A$  is translated to  $\text{Not}(A(?x)) \rightarrow C(?x)$ . Based on Assumption 3 (negation as failure), BCAR handles the translation rule as follows:

1.  $C(?)$  : returns the individuals which cannot be proved to be instances of  $A$ ;
2.  $C(? = a)$  : returns false if  $a$  is proved to be an instance of  $A$ , otherwise returns true;

- owl:allValuesFrom

$C = R \text{ only } A$  is translated to  $\forall ?y(R(?x, ?y) \wedge A(?y)) \rightarrow C(?x)$ . Based on Assumption 2, BCAR handles the translation rule as follows:

1.  $C(?)$  : returns the individuals which are proved to have some property values for  $R$  and all these values are proved to be instances to  $A$ ;
2.  $C(? = a)$  : returns true if  $a$  is proved to have some property values for  $R$  and all these values are proved to be instances of  $A$ , otherwise returns false;

- owl:minCardinality/maxCardinality/cardinality

$C = \text{min or max or exactly } a$  is translated to  $(R \geq \text{or } \leq \text{or } = a)(?x) \rightarrow c(?x)$ . Based on Assumption 2, BCAR handles the translation rule as follows:

1.  $C(?)$  : returns the individuals which are proved to have more than or less than or exactly  $a$  different property values for  $R$ ;

2.  $C(? = a)$  : returns true if  $a$  is proved to have more than or less than or exactly  $a$  property values for  $R$ , otherwise returns false;

- Complex ECA

A complex ECA is a combination of basic OWL constructs. As Table 4.1 shows, BCAR transforms complex ECA to multiple basic rules with self-created intermediate classes between them.

- `swrl:differentFrom/sameAs`

A SWRL rule may contain atoms such as “differentFrom( $?x,?y$ )” and “sameAs( $?x,?y$ )”.

In this case, BCAR firstly removes these atoms from TAL, before searching for solutions. After solutions are found, these comparison atoms are used to validate each solution based on Assumption 4. Only validated solutions are outputted in the end.

## 4.7 Experiments

As previously discussed, BCAR is developed majorally to address the ABox reasoning issues under CWA and UNA, which is demonstrated in Chapter 7 with a concrete case study. Besides that, in this section, several experiments are conducted to test the efficiency of the BCAR in searching solutions, proving that the general query performance of BCAR is reasonable compared with other general reasoners.

Table 4.2: Details of test ontologies

Ontology	Class	Property	Sub-class	Equiv.	Sub-Prop.	Domain	Range	C(a) (ABox)	R(a,b) (ABox)
<i>lubm_0</i>	43	25	36	6	5	25	18	18128	49336
<i>lubm_1</i>								40508	113463
<i>lubm_2</i>								58897	166682
<i>lubm_3</i>								83200	236514
<i>vicodi_0</i>	194	10	193	0	9	10	10	16942	36711
<i>vicodi_1</i>								33884	73422
<i>vicodi_2</i>								50826	110133
<i>vicodi_3</i>								67768	146844

#### 4.7.1 Test Ontologies and Queries

Two popular, well established ontologies, LUBM<sup>1</sup> and VICODI<sup>2</sup>, are chosen for the experiments. LUBM is a benchmark ontology developed at the Lehigh University for testing performance of ontology management and reasoning systems [40]. The ontology describes organizational structure of universities and it is relatively simple. LUBM comes with an ABox generator which generates ontologies (denoted as *lubm\_n*) with different sizes of ABox by setting the number of universities to  $n$ . VICODI is another ontology that is often used for evaluating the performance of ontology reasoners [90] [16]. The ontology is about European history, manually created in the EU-funded project VICODI. Originally VICODI ontology is written in RDFS, and has been translated to OWL in order to test BCAR. Different from LUBM which has its own ABox generator, for VICODI, a subset of ontologies are generated by duplicating original ABox axioms for  $n$  ( $n = 1, 2, 3, 4$ ) times, and are denoted as *vicodi\_n*. Details of the ontologies are listed in Table 4.2.

LUBM provides 14 standard queries for testing the ontologies. Four of them are

<sup>1</sup><http://swat.cse.lehigh.edu/projects/lubm/index.htm>

<sup>2</sup><http://www.vicodi.org>



selected in this study. The queries are written as:

*GraduateStudent*(?x)  $\wedge$

*takesCourse*(?x, *www.Department0.University0.edu/GraduateCourse0*)  $\rightarrow$  *Query01*(?x);

*GraduateStudent*(?x)  $\wedge$  *University*(?y)  $\wedge$  *Department*(?z)  $\wedge$  *memberOf*(?x, ?z)  $\wedge$

*subOrganizationOf*(?z, ?y)  $\wedge$  *undergraduateDegreeFrom*(?x, ?y)  $\rightarrow$  *Query02*(?x);

*Professor*(?x)  $\wedge$  *worksFor*(?x, *www.Department0.University0.edu*)  $\wedge$  *name*(?x, ?y1)  $\wedge$

*emailAddress*(?x, ?y2)  $\wedge$  *telephone*(?x, ?y3)  $\rightarrow$  *Query03*(?x);

*Student*(?x)  $\wedge$  *Faculty*(?y)  $\wedge$  *Course*(?z)  $\wedge$  *advisor*(?x, ?y)  $\wedge$  *teacherOf*(?y, ?z)  $\wedge$

*takesCourse*(?x, ?z)  $\rightarrow$  *Query04*(?x);

With above SWRL rules, in BCAR, generating results for query *Query01*(?), *Query02*(?), *Query03*(?) and *Query04*(?), is the process of searching for solutions for the rule body, which is essentially a conjunctive query of body atoms. In this way, BCAR is compared with other general reasoners that are available for conjunctive queries.

For VICODI ontologies, two queries adopted from [90] are used in the experiments, which are written in SWRL rules as well:

*Individual*(?x)  $\rightarrow$  *Query01*(?x);

*MilitaryPerson*(?x)  $\wedge$  *hasRole*(?y, ?x)  $\wedge$  *raleted*(?x, ?z)  $\rightarrow$  *Query02*(?x);

Please note that the above queries do not include any universal quantifier or exact cardinality inside the rule body, so that holding different world assumptions does not affect the reasoning results, otherwise BCAR is not comparable with other reasoners.

## 4.7.2 Experimental Results

The experiments compare BCAR with three other popular reasoners Pellet [117], RacerPro [41] and KAON2 [59]. The results are shown in Figure 4.5 (The bars that are over 1000000 ms line indicate time out).

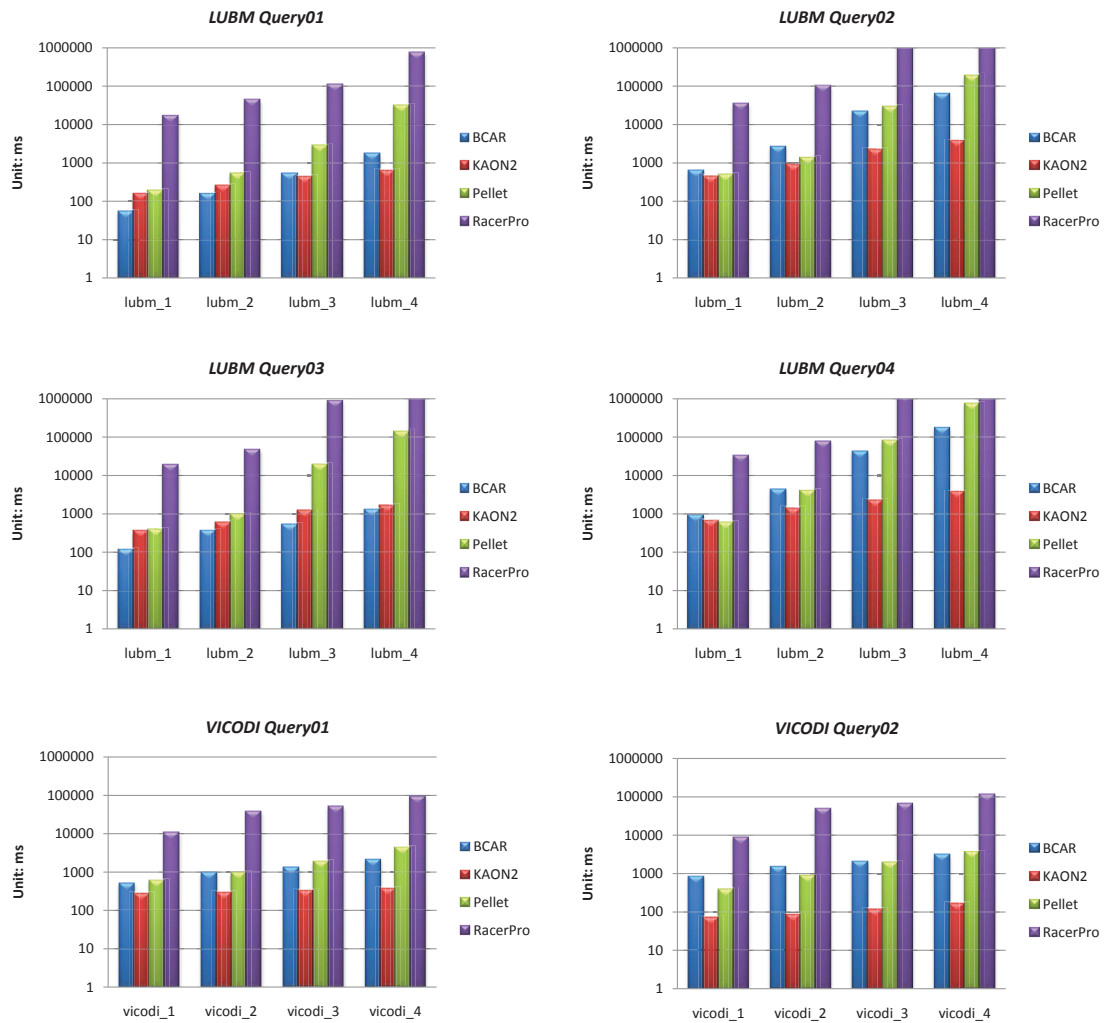


Figure 4.5: Experimental results

From the experimental results, compared with other reasoners, the performance of BCAR is decent. In most of the tests, BCAR can be ranked in the second place, which is slightly faster than Pellet, but is slower than KAON2 which is regarded as the fastest ontology reasoner recently [16]. In addition, BCAR performs very well for queries with high selectivity (e.g. *Query01* and *Query03* for LUBM) on relatively small datasets, due to the fact that the use of backward reasoning guarantees that only necessary information is processed.

Generally speaking, since that the major motivation of BCAR is to address the assumption mismatch issues, the general query performance is not significantly improved. In these experiments, BCAR shows a reasonable performance which ensures its usability.

## 4.8 Summary

This chapter has proposed a novel ABox Reasoner, BCAR, which is designed for ABox query reasoning, and can be used to help users discover instances or fillers for some defined classes or properties, defined using OWL ECAs and SWRL rules. The reasoner first translates ECAs to SWRL-like rules, and then backward reasons through the unified rule base to retrieve instances from the ABox of the ontology for the query tasks. OCQ technique has been introduced to improve the performance of reasoning. Experiments have shown that the BCAR has decent performance when compared with other general reasoners.

Automatically mapping a closed world based information model to an OWL ontology would definitely produce some reasoning problems, as they hold different assump-

tions. However, although the problems are obvious and the solution is not difficult, until now, there has not been any well-developed OWL reasoners that even mentions this problem. Researchers in knowledge engineering insist that OWL and SWRL are OWA-based languages without realizing that these languages have been used a lot for CWA-based information model. BCAR has filled this gap to a certain degree.

# Chapter 5

## Multiple Run Interactive Certainty Network

### 5.1 Introduction

This chapter proposes the core algorithm used for the SDM discussed in Section 3.6. SDM generates explanations for user observations of unexpected behaviors of a system, where AR is the most common cognitive model that has been used in this case. AR is a kind of logical inference, referring to the process of arriving at an explanatory hypothesis. Many computational AR models involve the process of generating hypotheses, evaluating hypotheses and selecting one of them as the best explanation. In this chapter, a novel probabilistic network, MRICN, which integrates human decision-making heuristics into a probabilistic approach, is proposed to perform interactive AR. MRICN is built upon ideas drawn from “Opinion Pooling”, “Probabilistic Network” and “Interaction”, allowing for reflective searching for the

optimal sets of knowledge with the maximal certainty gain. A simple automotive diagnostic experiment is used to test the network.

The rest of this chapter is organized as follows: Section 5.2 discusses the general ideas of interactive AR; Section 5.3 demonstrates how to construct the certainty network; Section 5.4 describes the algorithm for the interactive behavior; Section 5.5 gives an overview of the interactive reasoning process; Section 5.6 considers the implementation and experimental results with an automotive diagnostic case; and Section 5.7 summarizes the chapter.

## 5.2 Abductive Reasoning with Interactions

In MRICN, AR is used to provide diagnosis for the observed evidences, and the notion of interaction is introduced to the traditional AR to enhance adaptability of the system. As discussed in Section 3.6.2, interactions are defined as mutual influences between two coupled dynamic systems. The interaction between a computer system and a user is usually regarded as Human-Computer Interaction (HCI) that occurs at the user interface and often stands for a long term process of information sharing. Compared to the traditional input-output mapping systems that are memoryless and history-independent, interactive systems that provide history-dependent services can over time learn from and adapt to experience [142].

The ideas of interactive AR in this chapter can be summarized as follows:

1. The interaction between the system and the user is usually a multiple run process, indicating that the hypothesis generation using AR is a multiple run process as well.

2. The system should be able to identify the most important absence information, and can actively interact with the user in order to obtain the required information.
3. During the interaction, the state and behavior of the system should adapt to the dynamic context, which is updated upon the user's feedback for the system's questions.

### 5.3 Certainty Network Construction

This section discusses how the certainty network is constructed and how OP theory can be integrated. In most of the probabilistic causal networks, Bayesian theory is used for inference under uncertain circumstances. The serial connection can be solved by using the conditional probability method recursively. But the full-fledged joint probabilistic information for diverging connections is very hard to achieve. In MRICN, OP principle is applied to account for the diverging connection. Originally OP is used to aggregate experts' decisions [95] to support decision making in the presence of uncertainty. There are two major methods for OP: Linear Opinion Pool (LNOP) and Logarithmic Opinion Pool (LGOP). We are interested in LGOP because it is more likely to indicate consensual values when decisions must be made [35]. Supposing we have  $f(\theta)$  representing the overall distribution of probability of a proposition  $\theta$ , and  $f_i(\theta)$  representing the suggested probability distribution of the proposition  $\theta$  from each expert  $i$ , the equation for LGOP is:

$$f(\theta) = k \prod_i f_i(\theta)^{\omega_i} \quad (5.1)$$

where  $k$  is a normalizing constant that ensures  $f(\theta)$  integrates to 1, and  $\omega_i$  is the weight of each expert's opinion, which sums to 1. From equation (5.1), a further product of expert (POE) model was developed to provide sharper distribution without  $\omega_i$  [54]:

$$f(\theta) = k \prod_i f_i(\theta) \quad (5.2)$$

In Figure 5.1, a parent node  $A_1$  and a group of offspring nodes  $B_i (i = 1, 2, \dots, n)$  form a diverging connection (denoted as an offspring node  $B_i$  being caused by a parent node  $A_1$ ). We apply the Bayesian approach to account for the single connection between  $(A_1, B_i)$  independently. This can be viewed as the judgments from  $n$  experts in relation to the probability of  $A$  from the observations  $B_i$  respectively. Assuming  $B_i$  are independent factors and experts make judgment independently, the overall evaluation of the hypothesis  $A_1$  can be made by applying the OP approach to these judgments.

Considering the causal network shown in Figure 5.1, the conditional probability can be constructed from the statistic data set. The certainty of  $B_i$  is described as  $C(B_i)$ , and:

$$C(\overline{B_i}) = 1 - C(B_i) \quad (5.3)$$

where  $\overline{B_i}$  denotes that  $B_i$  is false.



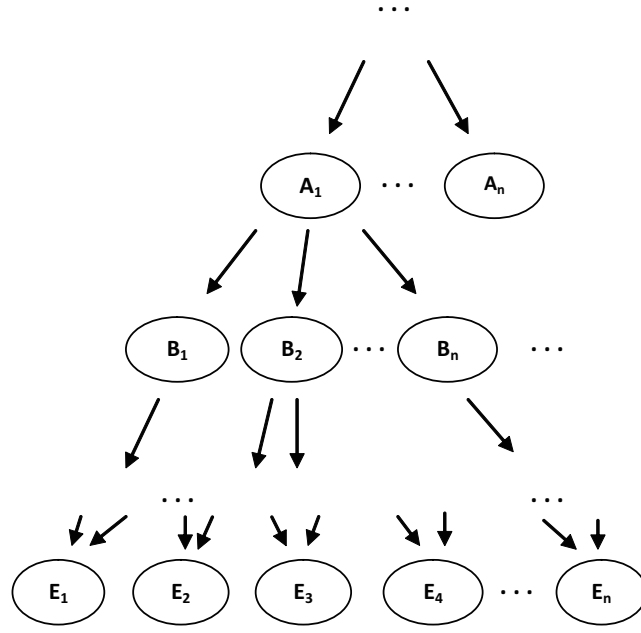


Figure 5.1: Causal network

Using the Bayesian formula, the certainty of  $A_1$  can be calculated using the data from each  $B_i$ :

$$\begin{aligned}
 C(A_{1(B_i)}) &= C(B_i, A_1) + C(\overline{B_i}, A_1) \\
 &= C(B_i) \times P(A_1|B_i) + C(\overline{B_i}) \times P(A_1|\overline{B_i})
 \end{aligned} \tag{5.4}$$

$$\begin{aligned}
 C(\overline{A_{1(B_i)}}) &= C(B_i, \overline{A_1}) + C(\overline{B_i}, \overline{A_1}) \\
 &= C(B_i) \times P(\overline{A_1}|B_i) + C(\overline{B_i}) \times P(\overline{A_1}|\overline{B_i}) = 1 - C(A_{1(B_i)})
 \end{aligned} \tag{5.5}$$

where  $C(A_{1(B_i)})$  indicates the independent calculation of certainty of  $A_1$  using the data of  $B_i$ .

Under the assumption that all the propositions are independent of each other, the OP equation (5.2) is applied, so that the certainty of  $A_1$  can be calculated as:

$$C(A_1) = \frac{\prod_i C(A_{1(B_i)})}{\prod_i C(A_{1(B_i)}) + \prod_i C(\overline{A_{1(B_i)}})} \quad (5.6)$$

$$C(\overline{A_1}) = \frac{\prod_i C(\overline{A_{1(B_i)}})}{\prod_i C(A_{1(B_i)}) + \prod_i C(\overline{A_{1(B_i)}})} \quad (5.7)$$

In this way, the certainty network can be constructed with only single conditional probability for each link.

## 5.4 An Algorithm for Interactive Behavior

Using the certainty network constructed above, the system can perform interactive reasoning. An interactive process is the process in which a system proactively pulls more environmental information (e.g. asks user questions) when it is in uncertain circumstances. For instance, after the system has calculated the certainties of all the hypotheses for a given evidence, if the hypothesis with the highest certainty is not valid in the interaction, the system requires additional environmental information as new evidence in order to update the network. To guarantee that the request is the most relevant one, the system searches for knowledge with the maximal fitness, which, in this case, is the evidence that achieves maximal certainty gain for the current activated hypothesis. Three steps are involved:

1. The system locates unknown evidences related to the activated hypothesis;
2. The certainty of each related unknown evidence is assumed to be 1 and the system calculates certainties of the activated hypothesis respectively. These certainties are called Alternative Certainty (AC);

3. Comparing ACs with one another, the system identifies the unknown evidence that provides maximal certainty gain.

The ACs are calculated using a recursive method. Let  $x, y$  denote certain nodes in the network,  $\psi_E$  denote the collection of all the unknown evidences,  $\psi_x$  denote the collection of unknown evidences related to  $x$ ,  $\phi_x$  represent the collection of offspring nodes of  $x$  (namely  $x$  is the cause of the members in  $\phi_x$ ) and  $ac_x$  is a mapping from the members in  $\psi_x$  to their ACs, then

$$\text{if } x \in \psi_E, \text{ then } \psi_x = \{x\}; \text{ else } \psi_x = \bigcup_{y \in \phi_x} \psi_y \quad (5.8)$$

and for each evidence  $e$  in  $\psi_x$  (assuming  $e$  comes from  $y_0$ , which is an offspring node of  $x$ ), applying the equation (5.6), the AC of  $x$  assuming  $e$  is true can be calculated as:

$$\begin{aligned} \text{if } x \in \psi_E, \text{ then } ac_x(e) = 1; \text{ else } ac_x(e) = \\ \frac{\prod_{y \in \phi_x, y \neq y_0} C(x_{(y)}) \times ac_{y_0}(e)}{\prod_{y \in \phi_x, y \neq y_0} C(x_{(y)}) \times ac_{y_0}(e) + \prod_{y \in \phi_x, y \neq y_0} C(\overline{x_{(y)}}) \times (1 - ac_{y_0}(e))} \end{aligned} \quad (5.9)$$

For example, as shown in Figure 5.1, supposing  $E_1$  and  $E_2$  are the unknown evidences related to  $B_1$ ,  $E_2$  and  $E_3$  are the unknown evidences related to  $B_2$ .  $B_1$  and  $B_2$  can simply pass their related evidences to their parent node  $A$ . In this way, the unknown evidences related to  $A$  are  $E_1, E_2, E_3$  respectively. Moreover, taking an unknown evidence, for example  $E_2$ , if we know the ACs of  $B_1$  and  $B_2$  assuming  $E_2$  is true, the AC of the parent node  $A$  can be calculated using equation (5.9). So when updating the network from bottom to top, both the certainty of a node and its

associated ACs can be calculated based on all of its offspring nodes. After MRICN is updated, each node in the network will then contain a value of its certainty as well as an array of its recorded ACs. The system searches the array of ACs of the current activated hypothesis to identify the unknown evidence with maximal certainty gain, which, in this case, is the difference between an AC and the certainty of the current activated hypothesis.

## 5.5 General Interactive Reasoning Process

The interactive reasoning process of the diagnosis using MRICN is shown in Figure 5.2, which involves 4 phases:

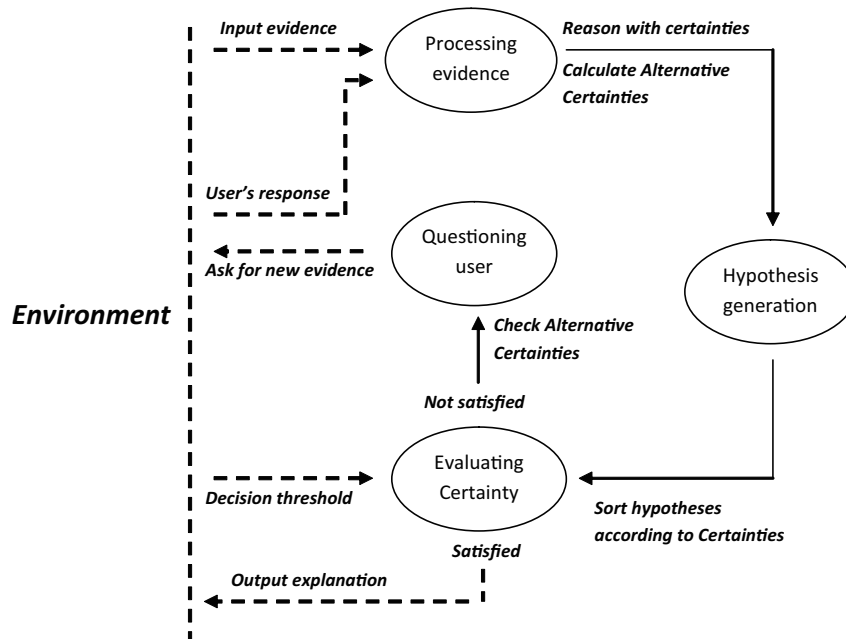


Figure 5.2: Interactive reasoning process

1. Processing evidence: Analyzing the input data and identifying known or unknown evidences;
2. Hypothesis generation: Reasoning using the equation (5.3)-(5.6) to generate hypotheses for the existing evidences. The hypotheses are sorted according to their certainties;
3. Evaluating certainty: Checking the certainties of hypotheses. The hypothesis with the highest certainty can be chosen as a plausible explanation if it is above the decision threshold. If none of the certainties is strong enough to support the decision making, the system needs to pull more evidence from the environment and re-assesses the certainties through the Questioning process;
4. Questioning: Identifying the unknown evidence that can provide maximal AC. The system interacts with the environment (a user in this case) to acquire the validity of the unknown evidences and re-assesses the circumstance by updating the certainty network, re-generating and re-ranking hypotheses accordingly. Moreover, if a system finds that the regenerated hypothesis remains invalid (i.e., the highest certainty is always lower than the decision threshold and there is no other unknown evidence provided for this hypothesis), the system runs a questioning process on other hypotheses with lower certainties.

## 5.6 Case Study

In this section, a scenario of automobile diagnosis is discussed to exemplify the MRICN model. The scenario is taken from [33], which discussed methods for provid-

ing explanations (or causes) for the observed symptoms of an automobile.

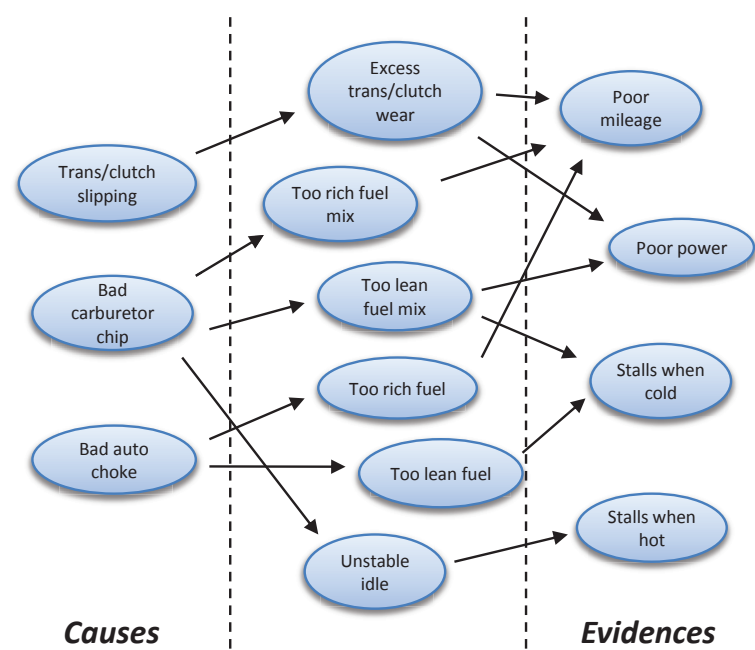


Figure 5.3: Instantiated certainty network

Figure 5.3 shows a portion of an instantiated certainty network. The nodes in the right panel represent the observed or unobserved evidences which need to be diagnosed, and the nodes in the left panel are causes that can be used to explain the observations. The nodes in the middle panel are intermediate propositions linking the causes to their potential effects. The links between nodes indicate probabilistic causal relationship, and the conditional probability for each link is shown in Table 5.1.

For example, probabilities linking a cause “Trans/Clutch Slipping” (TCS) to an intermediate proposition “Excess Trans/clutch Wear” (ETW) in the first line of Table 5.1 can be interpreted as:

Table 5.1: Empirical data adopted from an automobile fault diagnostic scenario

Causes (C)	P(C/I)	P( $\bar{C}$ /I)	Intermediate States (I)	P(I/S)	P( $\bar{I}$ /S)	Symptoms (S)
	P(C/ $\bar{I}$ )	P( $\bar{C}$ / $\bar{I}$ )		P(I/ $\bar{S}$ )	P( $\bar{I}$ / $\bar{S}$ )	
Trans/clutch slipping	0.95	0.05	Excess trans/clutch wear	0.3	0.7	Poor mileage
	0.05	0.95		0.1	0.9	
Trans/clutch slipping	0.95	0.05	Excess trans/clutch wear	0.9	0.1	Poor power
	0.05	0.95		0.4	0.6	
Bad carburetor chip	0.7	0.3	Too rich fuel mix	0.8	0.2	Poor mileage
	0.3	0.7		0.1	0.9	
Bad carburetor chip	0.7	0.3	Too lean fuel mix	0.8	0.2	Poor power
	0.3	0.7		0.3	0.7	
Bad carburetor chip	0.7	0.3	Too lean fuel mix	0.7	0.3	Stalls when cold
	0.3	0.7		0.5	0.5	
Bad carburetor chip	0.6	0.4	Unstable idle	0.9	0.1	Stalls when hot
	0.1	0.9		0.1	0.9	
Bad auto choke	0.7	0.3	Too rich fuel	0.7	0.3	Poor mileage
	0.1	0.9		0.5	0.5	
Bad auto choke	0.7	0.3	Too lean fuel	0.6	0.4	Stalls when cold
	0.1	0.9		0.2	0.8	

- if ETW is known to be true, the probability that TCS happens is 0.95;
- if ETW is known to be true, the probability that TCS does not happen is 0.05;
- if ETW is known to be false, the probability that TCS happens is 0.4;
- if ETW is known to be false, the probability that TCS does not happen is 0.6;

In this automobile diagnosis scenario, the system can be used to diagnose the failures of a vehicle within an interactive process. For example, a user may observe that his (or her) car has a poor mileage, and decides to use MRICN to find the reasons for the failure. Assuming that the user only accepts a recommendation with a certainty above a certain threshold (assuming 0.5 in this example), the functional process of MRICN is listed below (as shown in Figure 5.4):

1. After processing the evidence PM, the system identifies that TCS is the hypothesis with the highest certainty, in this case 0.4763, and the result is presented

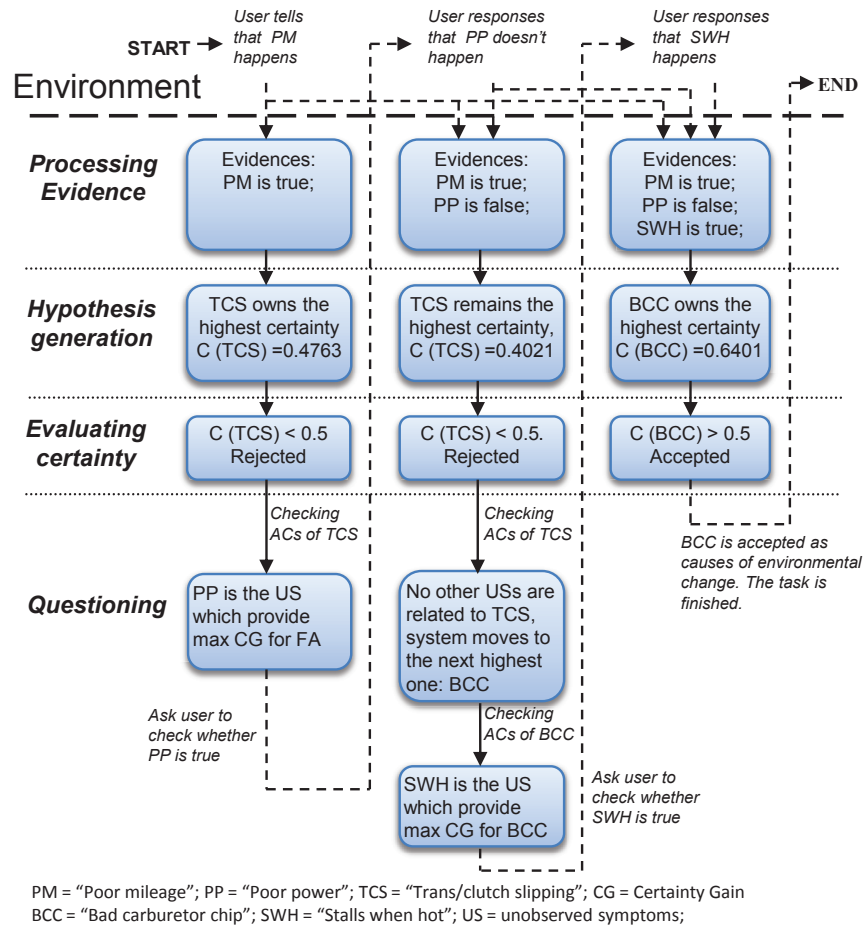


Figure 5.4: The functional process of MRICN



to the user;

2. Since the certainty of the hypothesis is lower than the assumed decision threshold 0.5, the user rejects this explanation. The system subsequently searches the evidences and infers that if PP is true, the certainty of the TCS would be maximally increased. The system then asks the user to validate the status of the new evidence PP;
3. If the user indicates that PP is false, the system initiates a new reasoning process based on this new observation, and indicates that TCS remains the highest certainty, this time 0.4021. If the system cannot identify any other unobserved evidences that can increase the certainty of the hypothesis TCS, it uses the next highest certainty hypothesis, in this example, BCC;
4. After assessing the array of ACs of BCC, the system reasons that if SWH is true, the certainty of BCC will be maximally increased. Consequently, the system initiates another interaction with the user to check whether SWH happens;
5. If we assume the user discovers that SWH happens, MRICN updates the network again and finds out that BCC has the highest certainty to cause this problem, which is 0.6401;
6. The user finally accepts BCC as the explanation since the certainty is now larger than the threshold, and the interactive reasoning process ends.

The inference flow of the above process is shown in Figure 5.5.

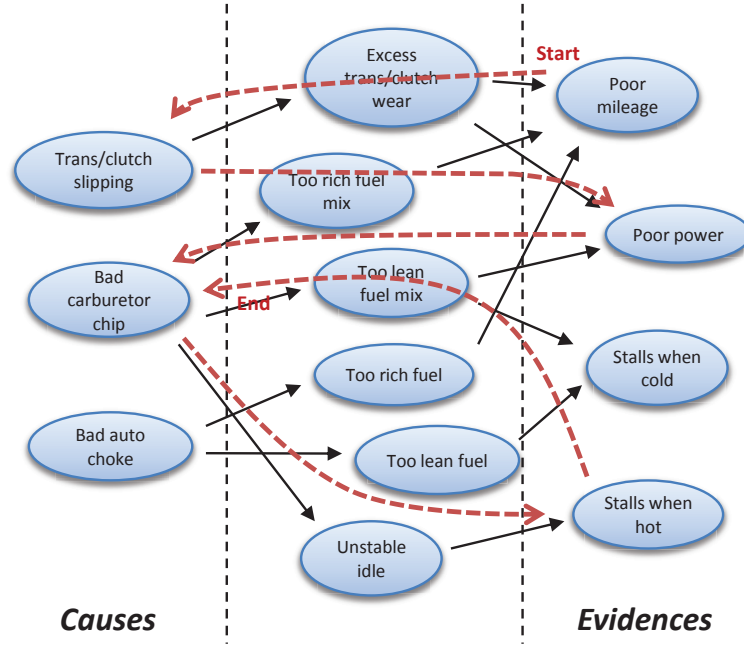


Figure 5.5: The inference flow

## 5.7 Summary

In this chapter, a progressive interactive certainty network, MRICN, has been proposed to generate explanations for observed evidences. The abduction theory underpinning MRICN is based on integrating OP theory into a probabilistic causal network. The fusion of Bayesian theory with OP enables MRICN to postulate a plausible explanation, despite the absence of detailed joint probability information. MRICN can interact with a user and seek unobserved evidence to maximize the fitness of its knowledge structures. A simple automobile fault diagnostic case has been adopted to test MRICN and some preliminary results have been produced.

MRICN is a key technology that has been developed for SDM. The idea comes from human-human medical diagnostic process which is essentially a multi-run interactive

process. In a real world medical diagnosis, the patient usually cannot at first identify all of the symptoms that have manifested as a result of his or her illness, but only the most obvious one. This preliminary checkup may not be enough for the doctor to correctly diagnose the illness. Multiple interactions are, thus, necessary for the doctor to draw new information, e.g. asking the patient to check whether some other relevant symptoms is true or not. MRICN has been developed to model this process.

This chapter only focuses on how to build MRICN and how to use MRICN to generate explanations. Issues regarding how to apply MRICN into software diagnosis and how to integrate the network with other techniques proposed in this study have not been discussed. In general, MRICN is the key technology used in SDM that helps users in providing explanations for the problems they encounter while using a KBS software. Details of integrating MRICN into the explanation model will be discussed in Chapter 7.

# Chapter 6

## Automated Feature Recognition

### 6.1 Introduction

AFR is a technology that automatically extracts high-level information, e.g. interesting geometrical patterns, from low-level shape representations for solid models of products or parts, so as to improve the efficiency of information sharing for collaborative design or Computer-Aided Design (CAD)/Computer-Aided Manufacturing(CAM) automation. The explanation framework proposed in this thesis is demonstrated through explanation production in AFR systems which act as KBSs that implicitly or explicitly apply feature-related knowledge to extract features from STEP-based solid models. STEP is the most popular data model for product information representation and exchange, which represents shapes based on Boundary Representation (B-rep). This chapter gives an overview of AFR as well as some background knowledge including B-rep and STEP.

The chapter is organized as follows: Section 6.2 gives an overview of AFR; Section

6.3 introduces B-rep which is the dominant approach for representing solid models; Section 6.4 briefly describes the STEP standard; Section 6.5 reviews existing AFR models; and Section 6.6 summarizes the chapter.

## **6.2 Overview of Automated Feature Recognition (AFR)**

### **6.2.1 Why AFR?**

Modern shape model design of products or parts relies heavily on Computer-Aided technologies (CAx). These systems are becoming highly collaborative and often involve multi-disciplinary project teams at distributed sites with heterogeneous computer systems. To share design information, existing product data formats (e.g. STEP, IGES, ACIS, etc.) can only describe basic geometric information, such as faces and edges, while the high level feature information which captures designer intent and manufacturing patterns is lost. Since increasingly intelligent CAx require the geometric model to be interpreted in terms of features [43] and manual feature recognition for growingly complex models with large data volume is impossible, AFR methods are developed. The features here can be explained as abstract concepts regarding some interesting geometrical or topological patterns. The role of AFR systems is extracting such information from low-level representations of shape models without interfering users.

Computer-Aided Process Planning (CAPP) is a typical application where AFR plays an important role. CAPP can be seen as a combination of CAD and CAM.

Given that CAD data of a part (a component of a product to be manufactured) is usually encoded using B-rep that only captures basic topological and geometrical information, the goal of CAPP is to generate a sequenced set of instructions used to manufacture the specified part [43]. In order to do that, CAPP has to interpret the part in terms of features, so that AFR systems are required to automatically extract features such as holes, slots and pockets, from the low-level representation of the part. CAPP then uses these features to generate manufacturing instructions in order to produce the part. For example, CAPP typically generates a drilling operation for a hole.



Figure 6.1: CNC machines

CAPP has been widely implemented as intelligent CAD/CAM systems for Computer Numeric Control (CNC) (as shown in Figure 6.1). A GA-based intelligent

CAD/CAM system for programming of CNC machine tools is proposed in [10], where AFR is the first step after CAD models are inputted. Figure 6.2 [10] shows the framework. Once features are recognized, the module for GA-based determination is taken over in order to determine: cutting tools, cutting parameters and detailed tool path planning. Afterwards post-processing takes place and encodes the tool-path data to the defined numerical control and machine tool.

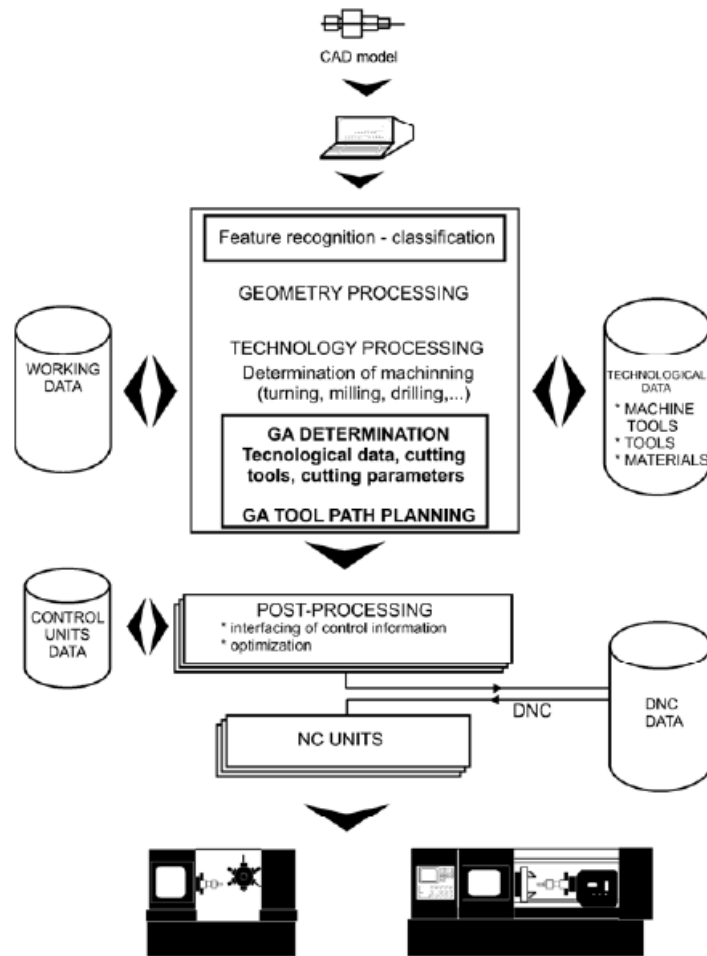


Figure 6.2: Concept framework of the GA-based intelligent CAD/CAM system

In addition, [62] proposed a CAPP approach manufacturing metallic aerospace components, where the importance of AFR has been highlighted. As is shown in Figure 6.3 [62], the original CAD model is converted to an IGES wire-frame model and curve group manipulation is carried out using the ICAD system. Rule-based algorithms are applied to groups of closed curves in order to identify fundamental manufacturing features of the product.

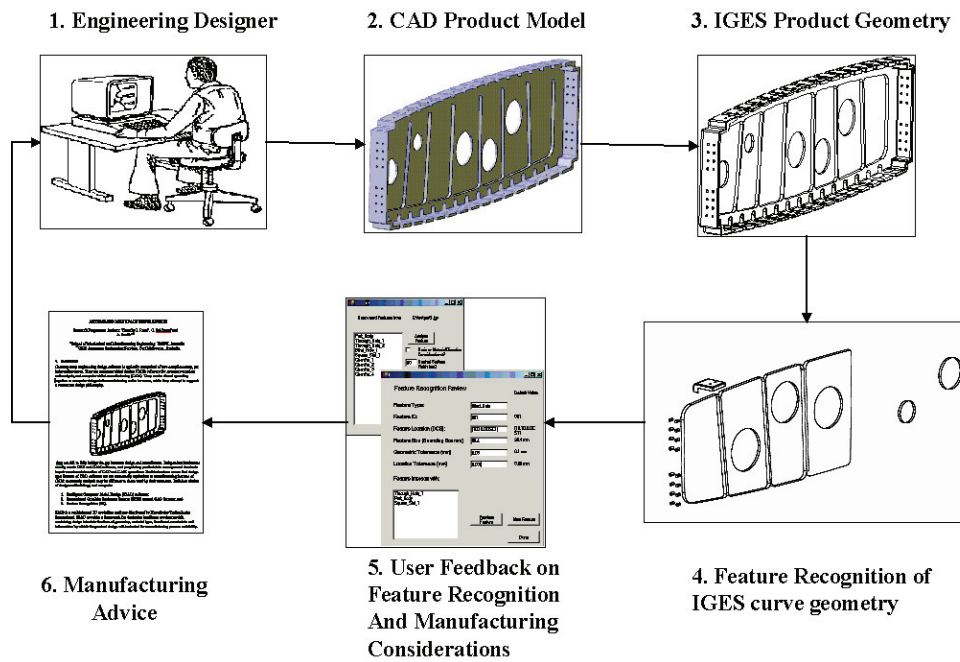


Figure 6.3: CAPP-based design cycle for metallic aerospace components

## 6.2.2 Major AFR Methods

The AFR research has attracted much attention. Many methods have been proposed to address this issue. In general, according to [43], there are three dominant



approaches in feature recognition:

1. graph-based algorithms: the graph-based approach creates a graph showing the topology of the product shape and then analyzes the graph to extract subsets of nodes and arcs that match any predefined template.
2. volumetric decomposition: two major methods are involved in this category: one is called convex hull volumetric decomposition, which repeatedly subtracts the input model from their convex hull in order to produce features; the other cell-based approach partitions a complex shape into simple convex shapes that are called cells.
3. hint-based approach: the hint-based algorithm recognizes features by identifying a set of relevant hints that represent the minimal indispensable portions of the feature.

Aside from the above three approaches, several other techniques in feature recognition have been identified in [9], including syntactic pattern recognition, state transition diagrams and automata, logic (if-then) rules and expert systems, and hybrid approach. Existing approaches vary greatly according to the recognition process, but Babic et al. [9] concluded that most of the AFR systems “apply a common basic principle: the structures identified in a part representation, formed using one of the above methods, are matched with some patterns in the knowledge-base, using if-then rules”. Therefore AFR systems are regarded as typical KBSs that have been used in the industry. However, although the explanatory facility has been considered as an important component for all KBSs, issues regarding explanation generation have

never been discussed in the existing AFR systems, which mostly focus on enhancing the accuracy or handling more complex models. The demonstration of the proposed explanation model with AFR applications fills this gap.

## 6.3 Boundary Representation of Solid Models

Most of the existing standards represent shape of products or parts based on B-rep, which is also a major application domain of many AFR systems. This section briefly introduces the basic ideas of B-rep.

B-rep is a popular method for representing the shape of solid models. In general, B-rep represents objects in terms of their “skin”. The skin is divided up into surface portions that are also called faces. The faces are surrounded by sequences of edges, which are portions of curves between two adjacent surface portions. Edges, or curve portions, are delimited by vertices, which are also meeting points for faces [120]. Figure 6.4 illustrates the relationships mentioned above.

The data structure of B-rep can be divided into two basic groups: one is responsible for defining the structure of the object (the topology) and the other is used to describe the form or shape of the object (the geometry). The main elements are the faces, edges and vertices, as mentioned above, together with their geometric forms: surfaces, curves and points. Other elements are also involved within the structure for various purposes, such as shell and loop. Figure 6.5 shows such basic data structure.

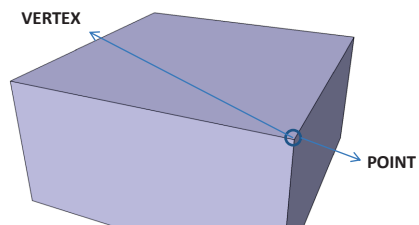
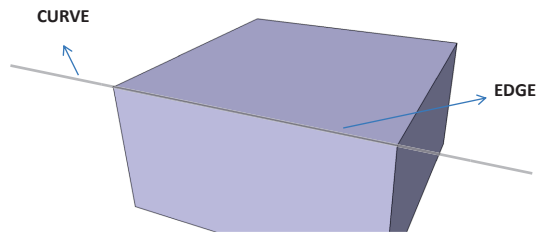
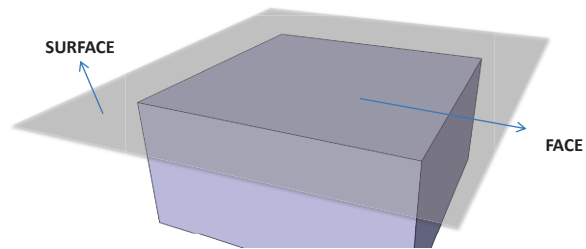


Figure 6.4: Face-surface, edge-curve, and vertex-point

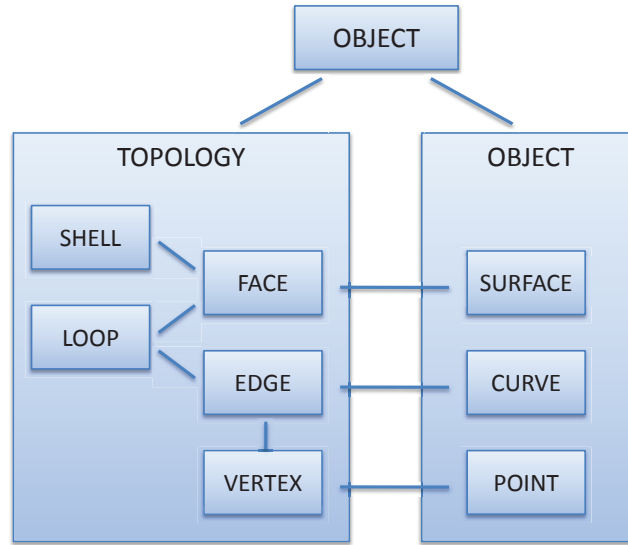


Figure 6.5: Basic data structure of B-rep

## 6.4 Standard for the Exchange Product Model Data (STEP)

Our explanation model is applied to the AFR systems that recognize features from STEP files. STEP is a very popular data model for product information representation and exchange. This section gives a general introduction of STEP.

STEP, also known as ISO 10303, is a top-rated data format for product exchange. The standard represents product information along the necessary mechanisms and definitions, which includes geometry, topology, tolerance, relations with other parts, various attributes and contingency to appropriate assembly [9], to enable product data to be exchanged among different computer systems and environments. Modern CAD is usually collaborative design which involves diverse computer systems and different organizations, so that STEP is developed to represent the product information in a

common computer-interpretable form that keeps the exchanged information complete and consistent [112].

### 6.4.1 Contents of STEP

The application range of ISO 10303 contains many different product types (e.g. electronic, mechanical, sheet metal and fiber composites) and life-cycle stages (e.g. design, analysis, planning and manufacture). A series of parts were issued continuously to expand the range. These parts are referred to as ISO 10303-xxx, where xxx is the part number. Each of them is a standard in its own right, and is independent on other parts [102]. In addition, STEP uses application protocols (APs) to specify the representation of product information for one or more application domains. APs are constructed on the basis of a set of Integrated Resources (IRs), defining fundamental constructs that can be specialized and applied for a wide variety of purposes [102]. Several important parts and APs are listed below:

- Parts:
  - Part 01: Overview and fundamental principles.
  - Part 11: EXPRESS language reference manual.
  - Part 21: Clear text encoding of the exchange structure.
  - Part 22: Standard data access interface specification.
  - Part 28: XML representation for EXPRESS-driven data.
  - Part 41: Fundamentals of product description and support.
  - Part 42: Geometric and topological representation.

- Part 43: Representation structures.
- Part 44: Products structure configuration.
- Application protocols:
  - AP 201: Explicit draughting.
  - AP 202: Associative draughting.
  - AP 203: Configuration-controlled 3D designs of mechanical parts and assemblies.
  - AP 207: Sheet metal die planning and design.
  - AP 210: Electronic assembly, interconnection and packaging design.
  - AP 214: Core data for automotive mechanical design process.
  - AP 224: Mechanical product definition for process planning using machining features.
  - AP 225: Building elements using explicit shape representation.
  - AP 238: Application interpreted model for computerized numerical controllers.

The part of STEP that is currently most widely being used is AP 203 “Configuration-controlled design”, which is also our focus in this case study. AP 203 represents product shape models as explicit non-parametric models based on B-rep (see Section 6.3) [102], and is used to exchange geometry, product structure, and configuration management data [112]. Many AFR systems have been proposed for recognizing features from AP203-based shape models [103]. This is also the case for the explanation framework proposed in this thesis.

### 6.4.2 EXPRESS

STEP primarily defines data models using the EXPRESS modeling language. EXPRESS is a standard data modeling language for product data that is defined in ISO 10303-11. Details of encoding mechanism that concerns data representation methods in accordance with a given EXPRESS schema are given in ISO 10303-21. Using EXPRESS, the main object created for a particular AP is called schema. Within a schema, sub-objects, including constants, types, entities, functions, procedures and rules, may be involved [111]. Details of some major objects are listed below:

- Schema: A schema defines a collection of objects that have a related meaning and purpose.
- Entity: An entity declaration creates a type that defines the properties of real-world or conceptual objects.
- Attribute: Attributes define the material properties of an entity and always have a value domain. There are two types of attribute: an explicit attribute is a property of an entity whose value is static and independent; a derived attribute is a property of an entity whose value changes in response to changes of other attribute values.
- Constant: A constant declaration is used to create values that never change.
- Type: A type specifies the natural type and meaning for data values. Built-in datatypes can be the common computational types (Boolean, integer, real, string, enumeration). Defined types are used to distinguish conceptually different collections of values that happen to have similar representations.

Generally, EXPRESS is used to represent the conceptual information within a data model (similar to TBox in an ontology), and a STEP file can be seen as an instantiation of the data model (similar to ABox in an ontology).

## 6.5 Existing AFR Systems

The AFR has long been realized as a key technology in automated design processes. Many research projects have been done to address this issue. The review mainly focuses on B-rep based geometric model, which have been the most commonly used [9].

According to the recognition principle, AFR can be mainly classified into three categories: graph-based approaches, volumetric decomposition approaches and hint-based approaches.

Graph-based theory was firstly proposed in 1988 [64], when the Attributed Adjacency Graph (AAG) had been developed. Once a B-rep model is transferred to an AAG, faces are represented as nodes, while the edges between the faces are represented as links between the nodes. In AAG, a link takes the attribute “0”, if the faces represented by the link’s nodes have a concave adjacency relation; otherwise, it takes the attribute “1”. A recognizer is then developed to recognize features by analyzing sub-graphs of the AAG. The original AAG concept suffered from two major shortcomings [9]: the possibility of application was only present for polyhedral objects without curved faces, and also the impossibility of extracting boundary faces (only basic faces). There were a series of later techniques developed to overcome these limitations, including Multi-Attributed Adjacency Graph (MAAG) [138], Structured



Face Adjacency Graph (SFAG) [30], etc. The two major drawbacks for all of these graph-based methods are their incapability of handling intersecting features and their having no guarantee that the recognized feature will prove applicable in the sense of manufacturability [43].

Volumetric decomposition methods decompose the geometric model into sub-volumes repeatedly until the feature is recognized. Convex hull based volumetric decomposition was originally introduced in 1991 to recognize form features as Alternating Sum of Volumes (ASV) decomposition [146]. In ASV decomposition, volumes of input models are decomposed by subtracting them from their convex hull and repeating the process for all the resulting volumes [15]. A critical problem of ASV decomposition is that the algorithm does not always converge. To solve this problem, the Alternating Sum of Volumes with Partitioning (ASVP) decomposition was developed by combining ASV and remedial partitioning using cutting operations [72]. Another direction volumetric decomposition is cell-based volumetric decomposition. The core idea of the cell-based approach is to partition a complex shape into simple convex shapes called cell. A specific algorithm for polyhedral objects was developed in [106], and is further extended to curved objects in [107].

Hint-based methods were developed with the motivation of overcoming the difficulties in handling the intersection. Hint-based approach was first implemented in Object-Oriented Feature Finder (OOFF) [137]. In OOFF, face patterns in a solid shape's B-rep generate hints or clues for the existence of certain machining features. For example, a cylindrical face could be a hint of a hole, and a pair of parallel opposing faces may be taken as a slot hint. These hints are tested for their validity through geometric completion procedures that attempt to construct the largest volumetric

feature that is consistent with the boundary data [44]. An advanced system, the integrated incremental feature finder (IF<sup>2</sup>), was further developed to extend OOFF by providing it with the ability to reason through the hints generated from various sources [44].

Another criterion to classify AFR systems is their recognition algorithm, according to which AFR can generally be grouped into two groups: rule-based approaches and ANN based approaches. In rule-based AFR systems, feature information is encompassed in a knowledge base using if-then rules. Features are recognized by applying inference mechanism to the rule base. Most of the rule-based AFR systems have been reviewed in [9]. The employment of ANN in feature recognition starts from early 1990s. In the system proposed in [101], an adjacency matrix is generated from a geometric model, which is similar to the AAG but includes more information. The matrix is then inputted into the neural network for recognition. Following works, including [80], [96] and [28] have later been developed to improve the performance of handling more complex model.

STEP files have been adopted as the input source by many AFR systems. In fact, most of the above methods have corresponding implementations on the STEP standard. For example, the implementation of IF<sup>2</sup> on STEP files was discussed in [42], and [66] gave a demonstration about how the graph-based approach could be used to recognize features from 3D sheet metal components that are represented by STEP AP-203. All other STEP-based AFR systems have different focuses. [78] proposed a modified graph-based approach that recognizes features from an auto-generated mid-surface adjacency graph, rather than the original adjacency graph; and another hybrid approach, which combines volume subtraction and face adjacency graph, was

put forward in [103]. For rule-based approaches, a STEP based AFR system was developed in [15], where the feature is recognized by applying if-then rules to face and edge information extracted from STEP files. Another interesting rule-based approach that is implemented using Prolog was designed in [154].

## 6.6 Summary

This chapter has introduced some background for AFR and other relevant knowledge, including the concepts of B-rep and STEP standard. In summary, STEP represents the shape of a product or a part based on B-rep, and STEP files have been widely adopted to represent and transfer product information. Hence, many AFR systems take STEP files as the source data, and focus on extracting features that were implicitly captured. We take such AFR systems as the case study in order to demonstrate our explanation framework, in so far as to propose an experimental Ontology-base AFR Explanatory System in the next chapter.

# Chapter 7

## Ontology-based AFR Explanatory System

### 7.1 Introduction

As discussed in Section 6.2, AFR systems are regarded as typical KBSs that explicitly or implicitly use rules to match the input shape models with some patterns in the knowledge base in order to make decisions, which, in this case, is to recognize features [9]. Therefore, it is reasonable to use AFR as a case study to demonstrate the explanation framework. To this end, a novel AFR explanatory system, OAES, which generates explanations for any AFR system used to recognize features from STEP-based (AP203) shape models, is outlined in this chapter.

The rest of this chapter is organized as follows: Section 7.2 gives an overview of OAES, including the concept framework; Section 7.3 describes the process of building the EKB in OAES; Section 7.4 demonstrates the usage of DEM; Section 7.5 discusses

the implementation of MRICN in OAES as SDM; Section 7.6 shows the user interface of the system; and Section 7.7 summarizes the chapter.

## 7.2 Overview of OAES

AFR systems extract features by automatically classifying groups of connected faces into different features, so that decision explanations, here, refer to the explanations of why a group of faces is recognized as a certain feature. OAES generates explanations using a reconstructive method. That is, explanations do not rely on the actual decision making process of the feature recognition, instead the explanatory system encodes perspicuous and well accepted geometric rules; and presents them to users, in order to explain how a feature is achieved. For example, it does not matter whether an AFR system uses the graph-based approach or the hint-based approach, ANN classification or logic rule matching, a pocket feature is always recognized based on its basic topology, namely a face intersects with all neighbor faces in concave angles. Different systems only differ in how the topology is represented and how the mapping of the topology into a pattern is conducted. For this reason, it is reasonable to expect a clear and understandable presentation of the topology to act as the explanation for why a group of faces is recognized as a certain feature.

Domain explanations in OAES focus on providing descriptive information for geometry entities and attributes defined in the EXPRESS schema of AP 203 in STEP. For example, they provide the definitions of “advanced face”, “edge curve”, “cartesian point” and how they are related. This information is captured statically in an ontology as RDF annotations of relevant classes or properties, and can be easily accessed

upon a user's request.

For software diagnostic, it is assumed in OAES that the system may prompt warnings or produce observable errors. MRICN (discussed in Chapter 5) is implemented to diagnose such problems. In general, MRICN first generates hypotheses for the problems that have been observed, then interacts with users to draw more situational information in order to evaluate the hypotheses, and finally selects one of the hypotheses as the best explanation.

According to the framework structure discussed in Chapter 3, the framework of OAES is shown in Figure 7.1. The system is independent to the AFR system, and generates explanations without referring to the actual feature recognition process executed in the AFR system. EKB is implemented as a geometric ontology that captures semantic rules and concepts for the domain knowledge. Once a STEP file is inputted into the AFR system for feature recognition, the ontology in OAES is instantiated by transferring the STEP file to the ABox of the ontology. Decision explanations, namely the explanations about how a feature is achieved, are produced by DEM, which uses BCAR to reason through the ABox of the ontology to logically connect the input (STEP file) and the output (recognized features). For domain explanation, detailed descriptions of objects in the domain, such as the definitions of “advanced face” and “edge curve” according to ISO10303, are embedded in the ontology as RDF annotations. In addition, MRICN is integrated in SDM with extra probabilistic knowledge, and provides explanations for user observations of software problems.

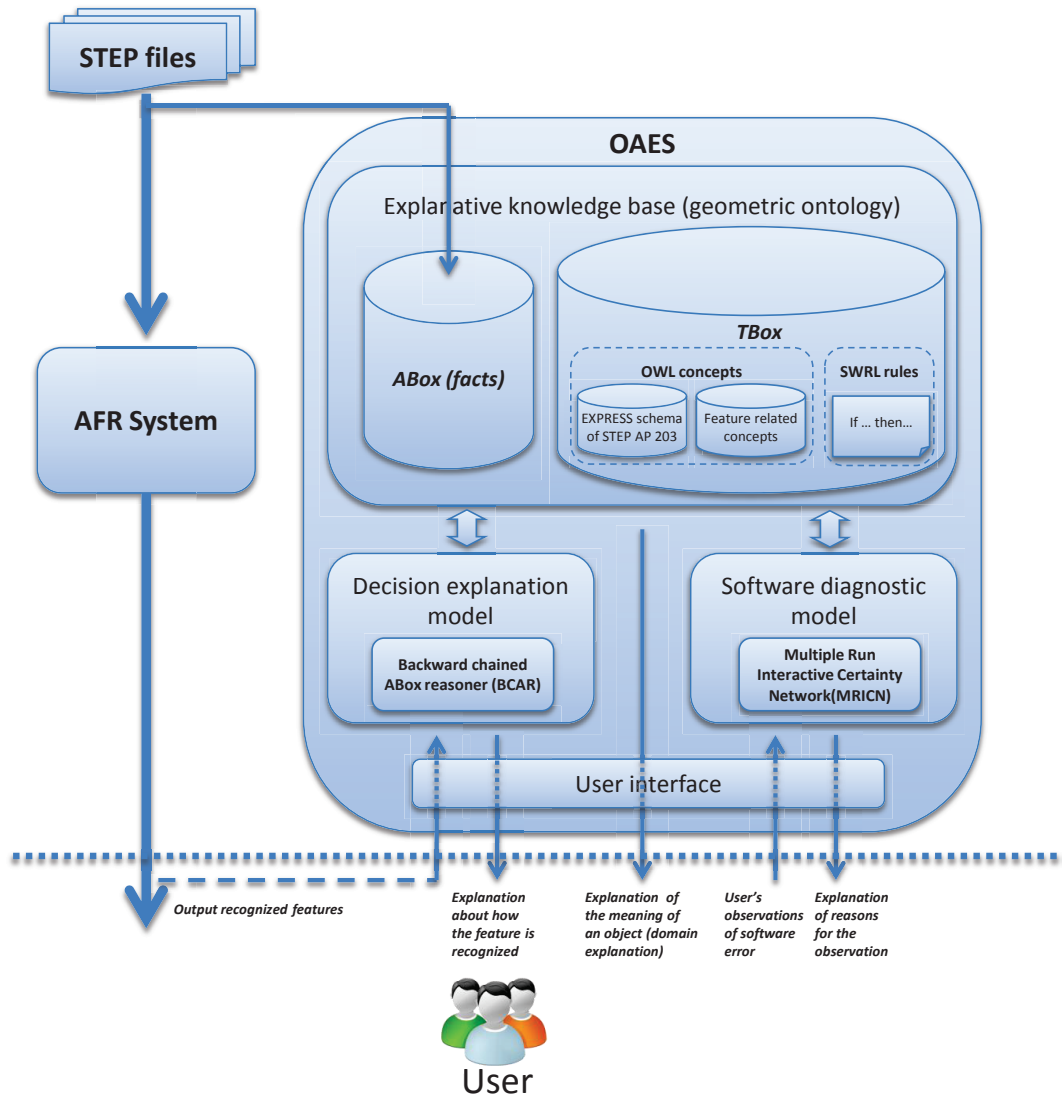


Figure 7.1: Ontology-based AFR explanatory system

## 7.3 Building Explanative Knowledge Base

The EKB in OAES refers to a geometric ontology, which aims to capture the domain knowledge regarding STEP AP203 and feature recognition, in a more perspicuous and explicit way. According to the four types of mapping discussed in Section 3.4.3, the development of the ontology consists of four steps:

1. Capturing EXPRESS schema of STEP AP203, which refers to the process of using ontology TBox to capture the domain concepts and relationships;
2. Defining feature recognition rules, which refers to the process of using SWRL rules or OWL ECA to capture the design rules and constraints;
3. Inserting additional descriptive information, which refers to the process of using RDF annotations to capture the definition and descriptions of the concepts and relationships within the domain;
4. Inputting STEP instances, which refers to the process of capturing input instances using ontology ABox;

### 7.3.1 Capturing EXPRESS Schema of STEP AP203

The first step of building the geometric ontology is to encode the concepts and relationships within the domain of STEP AP203 using OWL ontology.

As all the concepts and relationships involved in STEP AP203 are defined in its EXPRESS schema, the process of capturing domain concepts and relationships of AP203 is, in fact, the process of building an OWL ontology to map the EXPRESS



Table 7.1: Mapping from EXPRESS to OWL

EXPRESS	OWL
schema	ontology
entity type	class
super type/sub type	super class/sub class
explicit attribute of simple type	datatype property
explicit attribute of type entity	object property
entity name and entity attribute name	identifier for OWL representations

schema with all the elements included. Similar works have been done in [74] [156]. The general mapping information can be found in Table 7.1.

The complete AP203 includes many constructs which are concerned with different aspects of information in product design and configuration, such as organizations, date, approvals, security, classifications and etc. Here we only focus on the portion which represents product shapes: the *representation\_item* entity and all its subtype entities. According to B-rep, the *representation\_item* entity has two direct subtype entities: *geometric\_representation\_item* and *topological\_representation\_item*. A portion of the hierarchy of subtype entities of *representation\_item* is shown in Figure 7.2.

In the following discussion, the concepts and relationships involved in the EXPRESS schema of AP203 are called Schema-Based Resource (SBR), and their corresponding classes and properties are called Schema-Based Classes (SBC) and Schema-Based Properties (SBP) respectively.

### 7.3.2 Defining Feature Recognition Rules

To explain why a group of faces is recognized as a feature, the system outputs intelligible recognition rules representing the topology of the face group to the users.

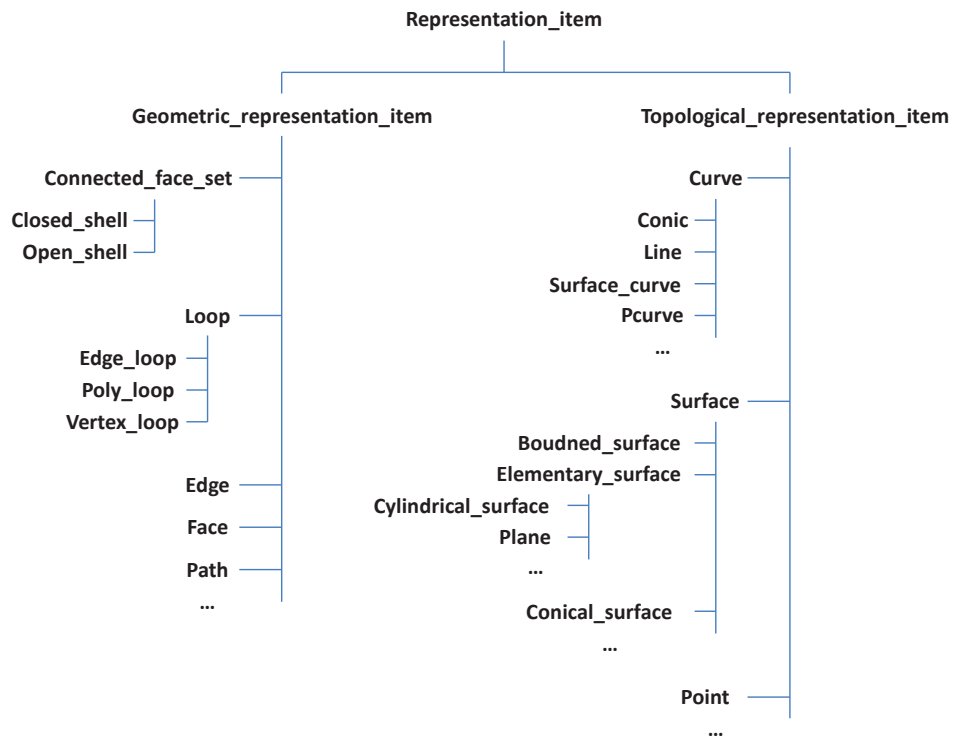


Figure 7.2: Portion of subtype entity hierarchy of *representation\_item*

Taking the triangular pocket (shown in Figure 7.3) as an example, a group of four faces is recognized as a triangular pocket because one of them is recognized as a pocket bottom face, the other three are recognized as pocket wall faces, and they are concavely connected with each other. Moreover, a face is recognized as the pocket bottom face because it intersects with all its neighbor faces in concave angles. Figure 7.4 shows all the available features that can be explained by OAES. The set of the feature is limited by the developed sets of rules. The list of the rules for those features can be found in Appendix A.

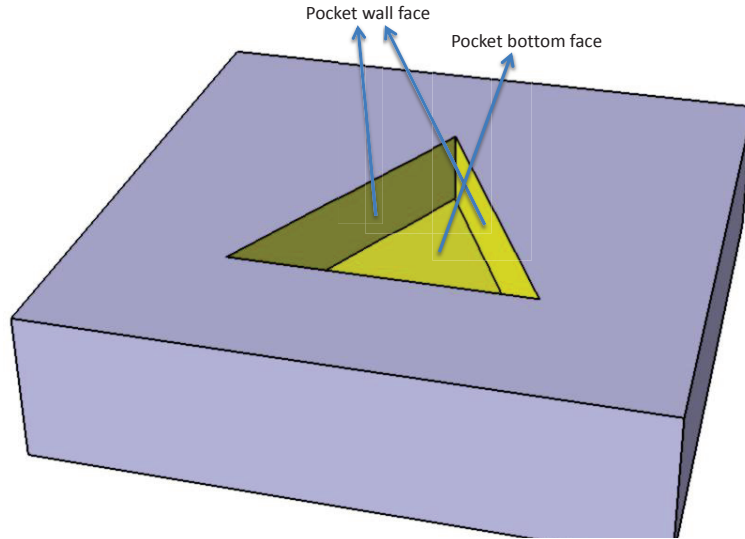


Figure 7.3: Example of a triangular pocket

There are two methods for building recognition rules: SWRL and OWL ECA. The atoms (excluding comparison atoms and Built-in function atoms) in a SWRL rule must have corresponding classes or properties in the OWL ontology. However the recognition rules usually include some atoms that represent feature-related con-

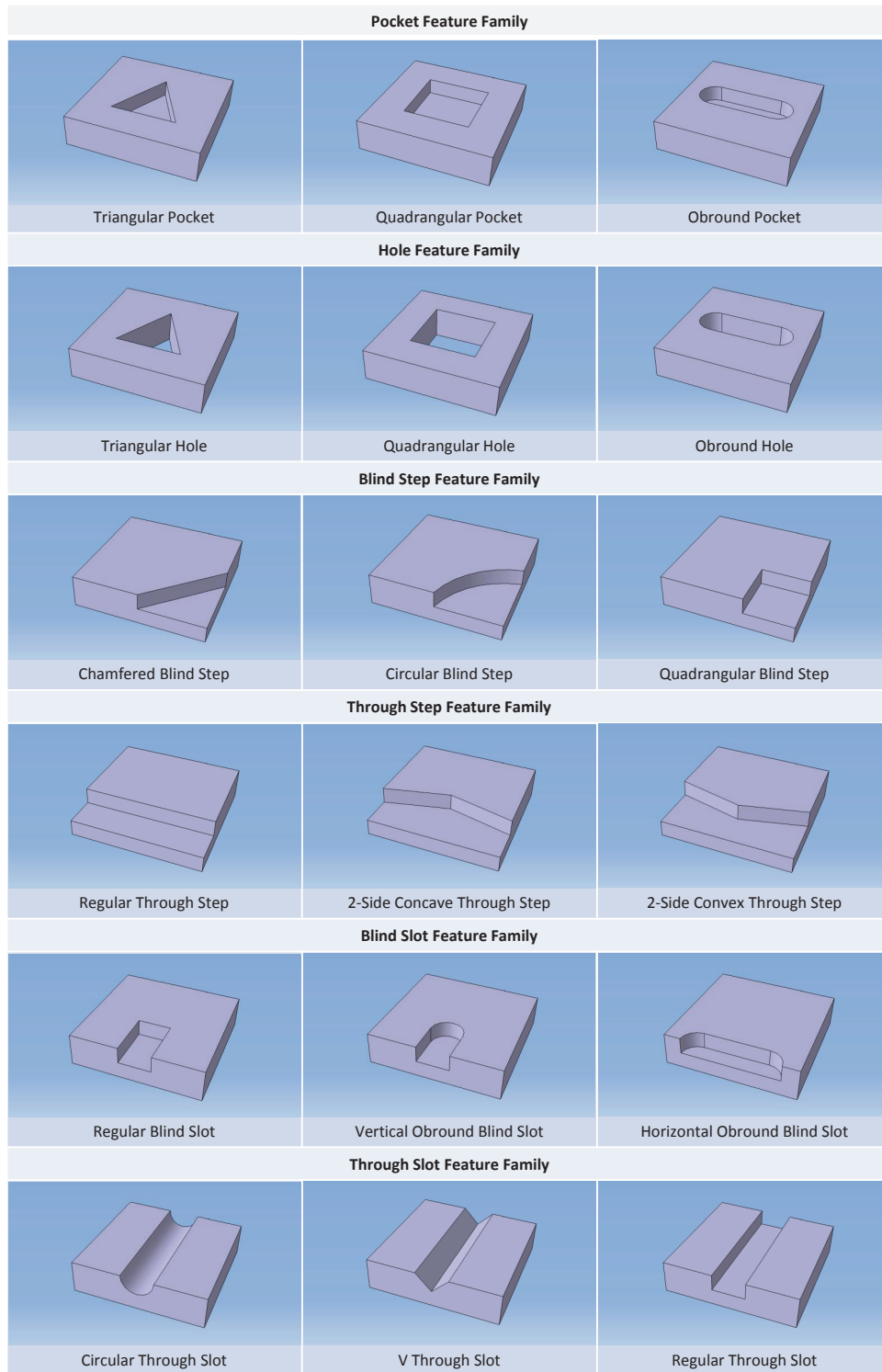


Figure 7.4: Explainable features in OAES

cepts or relationships which are not SBRs and do not have corresponding SBCs or SBPs. Thus these OWL classes and properties need to be created before SWRL rules are constructed. In the following discussion, these feature related concepts and relationships are called Feature-Related Resources (FRR), and their corresponding classes and properties are called Feature-Related Classes (FRC) and Feature-Related Properties (FRP) respectively.

In OAES, the recognition rule for a feature is also regarded as the definition of the feature. Namely, in order to explain how a feature is recognized, OAES presents the user with how the facts match the definition, so as to deduce the feature. The general process of developing the recognition rules is as follows:

1. Other than the *representation\_item* class that covers all the SBCs, classes for features are created to be the subclasses of another top class *face\_set*, indicating that instances of features are essentially sets of faces.
2. After the creation of classes that represent features, for each feature class, the face topology of the feature is analyzed, and is abstracted as if...then... rules for recognizing the feature.
3. The if...then... rules may contain FRRs that have not been mentioned in the ontology, so that corresponding FRCs and FRPs are further created for them.
4. These rules are, then, transferred to SWRL or OWL ECA involving the newly created FRCs and FRPs.
5. The newly created FRCs and FRPs are further defined using rules similar to the way described from the step 2 to 4. More undefined FRRs may be involved.

Repeat this process until all the FRRs involved are defined by rules.

A triangular pocket example is discussed here to demonstrate the above process. An OWL class *triangular\_pocket* is first created as a subclass of *face\_set* according to step 1. As shown in Figure 7.3, a *triangular\_pocket* consists of one bottom face and three wall faces, which are neighbor faces of the bottom face (step 2). Corresponding OWL classes (*triangular\_pocket\_bottom\_face* and *pocket\_wall\_face*) and properties (*face\_has\_neighbor*) are created according to step 3. The SWRL recognition rule for *triangular\_pocket* is then written as (step 4):

$$\begin{aligned} & \text{face\_set}(?x) \wedge \text{has\_member\_face}(?x, ?a) \wedge \text{has\_member\_face}(?x, ?b) \wedge \text{has\_member\_face}(?x, ?c) \wedge \\ & \quad \text{has\_member\_face}(?x, ?d) \wedge \text{triangular\_pocket\_bottom\_face}(?a) \wedge \text{pocket\_wall\_face}(?b) \wedge \\ & \quad \text{pocket\_wall\_face}(?c) \wedge \text{pocket\_wall\_face}(?d) \wedge \text{face\_has\_neighbor}(?a, ?b) \wedge \\ & \quad \text{face\_has\_neighbor}(?a, ?c) \wedge \text{face\_has\_neighbor}(?a, ?d) \wedge \text{face\_has\_neighbor}(?b, ?c) \wedge \\ & \quad \text{face\_has\_neighbor}(?c, ?d) \wedge \text{face\_has\_neighbor}(?d, ?b) \rightarrow \text{triangular\_pocket}(?x) \end{aligned}$$

The above rule simply means: if a *face\_set* (*?x*) contains one *triangular\_pocket\_bottom\_face* (*?a*) and three *pocket\_wall\_faces* (*?b, ?c, ?d*), and the three *pocket\_neighbor\_faces* are circularly connected with each other and are all connected with the bottom face *?a*, then the *face\_set* (*?x*) is a triangular pocket. Seeing as the OWL class *triangular\_pocket\_bottom\_face*, *pocket\_wall\_face* and the OWL property *face\_has\_neighbor* are FRCs and FRPs which do not have corresponding entities or attributes in the EXPRESS schema of AP203, they need to be further defined (step 5) as:

$$\begin{aligned} & \text{face} \sqcap \exists \text{face\_has\_neighbor.pocket\_bottom\_face} \equiv \text{pocket\_wall\_face}; \\ & \text{triangular\_face}(?x) \wedge \text{pocket\_bottom\_face}(?x) \rightarrow \text{triangular\_pocket\_bottom\_face}(?x) \\ & \text{face}(?x) \wedge \text{face}(?y) \wedge \text{face\_has\_edge}(?x, ?e) \wedge \text{face\_has\_edge}(?y, ?e) \wedge \text{differentFrom}(?x, ?y) \rightarrow \\ & \quad \text{face\_has\_neighbor}; \end{aligned}$$

Table 7.2: Mapping between OWL and EXPRESS items within the *triangular\_pocket* example

Express entities or attributes	OWL classes or properties
face (entity)	face (class)
plane(entity)	plane (class)
face_geometry (attribute of "face_surface" entity)	face_surface_has_face_geometry (object property)
bounds (attribute of "face" entity)	face_has_face_bounds (object property)
bound (attribute of "face_bound" entity)	face_bound_has_bound (object property)
edge_list (attribute of "edge_loop" entity)	edge_loop_has_edges (object property)
edge_element (attribute of "oriented_edge" entity)	oriented_edge_has_edge_element (object property)

The OWL class *pocket\_wall\_face* is defined through an OWL ECA, which means that a *pocket\_wall\_face* is a neighbor face of a *pocket\_bottom\_face*. Meanwhile, the second SWRL rule states that a *triangular\_pocket\_bottom\_face* is a *triangular\_face* as well as a *pocket\_bottom\_face*. The OWL property, *face\_has\_neighbor* is defined using a SWRL rule as well, indicating that two different *faces* (*?x, ?y*) are neighbors if they have a same *edge* (*?e*). In general, it is preferable to use SWRL, rather than OWL ECA, to construct rules, unless some special logic is required, such as universal quantification or cardinality restriction, which are not or partly supported in SWRL. In addition, the above rules bring more OWL items that have not yet been defined, including *triangular\_face*, *pocket\_bottom\_face* and *face\_has\_edge*, so that more rules are added.

In such way, all the FRCs and FRPs involved for recognizing the feature *triangular\_pocket* are now defined in above rules. Other atoms involved in these rules are all SBRs. The creation of their corresponding SBCs and SBPs is discussed in the previous section, and is shown in Table 7.2. Generally speaking, these recognition rules logically link the features to the facts stated in the STEP file, allowing the expla-

nations, regarding how a feature is recognized from a STEP file, to be provided by presenting the rules to the user in a clear and descriptive way. How to present these rules as explanations is discussed in Section 7.4.

### 7.3.3 Inserting Additional Descriptive Information

OWL is an ontology description language developed on top of RDF, so that all the OWL items are essentially RDF resources. RDF allows additional support information to be caught by a set of utility properties for each resource, including `rdfs:seeAlso`, `rdfs:isDefinedBy`, `rdfs:comment` and `rdfs:label`. Using this method, OAES captures additional descriptive information for generating domain explanations and polishing decision explanations. The information includes: 1) descriptions of SBRs; 2) descriptions of FRRs; 3) readable format of recognition rules for decision explanations; All the three types of information are encoded as HTML text, which provides many benefits for the presentation of the information, e.g. highlighting important text, or integrating hyperlink to allow for user-system interaction.

For SBRs, descriptive information regarding their meaning and their attributes is captured using the “`rdfs:comment`” property of the corresponding SBCs and SBPs. This information is considered as domain explanations for the SBRs, which can be directly provided to the user upon request. The information comes from ISO/CD 10303-42: 1992, and can be found in <http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifcgeometryresource/lexical/>. The URL link for each concept is also mentioned in the value of the property “`rdfs:seeAlso`”. Figure 7.5 is an example showing the descriptive information of the OWL class of *edge*.





Property	Value
 <code>rdfs:comment</code>	<p>An edge is the topological construct corresponding to the connection of two vertices. More abstractly, it may stand for a logical relationship between two vertices. The domain of an edge, if present, is a finite, non-self-intersecting open curve in RM, that is, a connected 1-dimensional manifold. The bounds of an edge are two vertices, which need not be distinct. The edge is oriented by choosing its traversal direction to run from the first to the second vertex. If the two vertices are the same, the edge is a self loop. The domain of the edge does not include its bounds, and <math>0 \leq \Xi \leq \infty</math>. Associated with an edge may be a geometric curve to locate the edge in a coordinate space; this is represented by the edge curve (<code>IfcEdgeCurve</code>) subtype.</p> <p>Attributes:  EdgeStart -&gt; Start point (vertex) of the edge.  EdgeEnd -&gt; End point (vertex) of the edge. The same vertex can be used for both EdgeStart and EdgeEnd.</p>
 <code>rdfs:seeAlso</code>	<a href="http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifctopologyresource/lexical/ifcedge.htm">http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifctopologyresource/lexical/ifcedge.htm</a>

Figure 7.5: Descriptive information of *edge*

For FRRs which do not have corresponding entities in the STEP AP203 schema nor formal definitions, their descriptions are generally based on their topology patterns, and are also captured using “`rdfs:comment`” property. The FRRs do not have the “`rdfs:seeAlso`” property, as they do not have other resources explaining them. Instead, “`rdfs:isDefinedBy`” property gives a verbal description of the rules defining them. Two restrictions need to be applied when encoding the contents: 1) All the terms representing basic SBRs must be encoded as HTML links which allow the user to ask for domain explanations for the involved SBRs by clicking the links; 2) The verbal description of a definition rule must contains all the variables that are included in the definition rule. The HTML text of descriptive information of *concave\_angular\_edge* is shown in Figure 7.6 as an example.

### 7.3.4 Inputting STEP Instances

The process of inputting STEP instances refers to the process of instantiating the ontology TBox by inputting facts. OAES creates OWL individuals to map instances in the STEP files. OWL individuals represent objects in the domain of discourse, and



Property	Value
 rdfs:comment	A concave_angular_edge is an <a href=edge>edge</a> whose two adjacent <a href=face>face</a> intersects at an concave angle. The edge is defined by an SWRL rule which contains a function that calculates angles between two faces.
 rdfs:isDefinedBy	?x is a <a href = concave_angular_edge>concave_angular_edge</a> because ?x is an <a href=edge>edge</a> of two different <a href=face>faces</a> ?y and ?z, and the intersection angle between ?y and ?z is smaller then 180 degree,.

Figure 7.6: Descriptive information of *concave\_angular\_edge*

can be referred to as “instances of concepts” [56].

An entity in the EXPRESS schema of AP203 refers to a concept in the domain; and a STEP file, as an instantiation of the EXPRESS schema, is a collection of the instances of the EXPRESS entities. Once the system reads an instance in the STEP file, it first locates the corresponding OWL class of the EXPRESS entity that the instance belongs to, then creates an OWL individual as a member of the OWL class, and finally relates the individual to other existing individuals using OWL properties based on the EXPRESS attributes of the corresponding STEP instance.

For example, an instance in STEP file is stated as:

$$\#245 = LINE('Line', \#242, \#244)$$

The name of the corresponding OWL individual is ID245, and the individual belongs to a concept *line*. Assuming there are two properties related to the concept “line”: *line\_has\_point* and *line\_has\_direction*, the values of these two properties of individual ID245 are individual ID242 and ID244 respectively.

OAES has implemented JSDAI technology to help the input of STEP instances. JSDAI is a Java-based Application Programming Interface (API) for reading, writing and runtime manipulation of object oriented data defined by an EXPRESS based

data model <sup>1</sup>. The employment of JSDAI core libraries allows the system to access STEP models and manipulate STEP data. In OAES, the system first uses JSDAI-based functions to read the STEP instances in the system memory, then creates OWL individuals for the instances in a certain sequence.

## 7.4 Decision Explanation Model

In OAES, decision explanations are the explanations of why a group of faces is recognized as a certain feature. OAES generates explanations based on a set of recognition rules showing the topology of the feature. Determining which recognition rules are appropriate for a particular explanation task is essentially a reasoning issue, namely to find out which recognition rules should be applied to recognize a particular feature from a STEP file. Since Chapter 3 has given a general introduction of DEM and the technical details of BCAR have been discussed in Chapter 4. This section mainly focuses on how BCAR is used in OAES and how to generate explanations in natural language. A simple example is then discussed to demonstrate the entire process.

### 7.4.1 Generating Decision Explanations

BCAR is a general rule-based ontology ABox reasoner. OAES applies BCAR to reason through the ontology and locate the recognition rules explaining the recognition process. Once a user wants to know why a set of faces has been recognized as a certain feature, he or she may input these faces into OAES for explanation. OAES first creates a new OWL individual under the OWL class *face\_set*. The newly

---

<sup>1</sup><http://www.jsdai.com>

created individual is then related to the inputted faces through an OWL property *has\_member\_face*, indicating that the newly created individual is a *face\_set* which contains such member faces. The next process involves using recognition rules to justify why the newly created OWL individual can be classified as the recognized feature, and presenting the readable format of the rules to the user as an explanation. The general process is as follows (let  $f$  denote the feature and  $i$  denote the newly created individual):

1. The system firstly initializes BCAR and sets the reasoning task as  $f(? = i)$ , in other words, checking whether  $i$  is an instance of  $f$ .
2. After BCAR has completed the reasoning process, if it returns false, meaning OAES cannot verify that  $i$  is an instance of  $f$ , the system then asks the user to recheck the input. Otherwise if BCAR returns true, OAES proceeds to the next step.
3. Combined with the reasoning solution concluded by BCAR, the primary rule, which directly states that  $i$  belongs to  $f$ , is first presented to the user.
4. If the user requests further explanation about the conditions in the primary rule, rules which have verified those conditions are then presented to the user as a further explanation.
5. Repeat step 4 until all the recognition rules involved in this task are presented or the user terminates the interaction.

In the above process, rules are presented in natural language. To this end, the verbal description, which is captured by “*rdfs: isDefinedBy*” property (discussed in

Section 7.3.3), is used. BCAR solves the reasoning problems by obtaining solutions for the rules involved (discussed in Chapter 4). In a solution, every variable in the rule is bound to a value, making the rule conditions hold true. Since that the verbal description of a rule contains all the rule variables, in order to generate explanations, OAES finally outputs the rule’s verbal description with all the variables replaced by the binding values in the solution. In addition, as a support to the natural language representation, the original format of the rule is attached to the presentation with all the variables replaced by the binding values as well.

All the explanations are presented as HTML texts, which contain HTML hyperlinks enabling a simple interaction between the user and the system. The hyperlinks are used for three purposes: 1) referring to domain explanations; 2) referring to further explanations of rule conditions; 3) referring to asserted facts. In this case, OAES does not provide all the explanations at one time, as it contains too much information and is difficult for the user to follow. Instead, the explanations are provided one by one upon the user’s request, linking the feature to the asserted facts. Details of the use of hyperlinks are demonstrated in the next subsection with a simple example.

## 7.4.2 Demonstration Example

A simple example is discussed in this section to demonstrate the process mentioned in the previous section. Let us assume that four faces, *ID001*, *ID002*, *ID003* and *ID004* (corresponding to the STEP instances #001, #002, #003, #004), have been recognized as faces that constitute a triangular pocket as shown in Figure 7.3. An OWL individual *F001* is then created as a *face\_set*, and is related to *ID001*, *ID002*,

$ID003$  and  $ID004$  with the *has\_member\_face* property. The reasoning task is set to *triangular\_pocket*( $? = F001$ ).

Assuming the reasoner returns that  $F001$  has been proved to be a *triangular\_pocket* using the recognition rules, and a solution for the primary rule has been found as “ $\{?x \leftarrow F001; ?a \leftarrow ID001; ?b \leftarrow ID002; ?c \leftarrow ID003; ?d \leftarrow ID004\}$ ”, OAES then presents a sequence of explanations as shown in Figure 7.7.



Figure 7.7: A sequence of explanations for “why  $F001$  is a *triangular\_pocket*”

In the explanation sequence, the readable format of the primary rule, with all the

variables replaced by the binding values, is first presented as the explanation for why *F001* is recognized as a *triangular\_pocket* (as shown in the top of Figure 7.7). Within this first explanation, as the blue arrow shows, the user can click the HTML hyperlinks to request for domain explanations regarding the concepts mentioned in the explanation. For example, if the user wants to know what a *triangular\_pocket\_bottom\_face* is, the domain explanation is provided by directly accessing the descriptive information of *triangular\_pocket\_bottom\_face*, captured by the “*rdfs:comment*” property.

Moreover, as the green arrows show, by clicking the HTML hyperlink at the end of the explanation, the user can also get the original SWRL format of the primary rule. Each hyperlink in the original SWRL rule refers to a condition atom, which must hold true in order to prove that *F001* is a *triangular\_pocket*. By clicking a hyperlink, the user is able to know why the corresponding condition holds true. There are two alternatives in this case: 1) the condition atom is true because it is an asserted fact, e.g. *has\_member\_face(F001, ID001)* (as the black arrow shows); 2) the truth of the condition atom is explained by other rules, e.g. *pocket\_wall\_face(ID001)* (as the red arrow shows).

In this way, the user can keep clicking the HTML links within the previous explanation if they need further detailed explanations, until all the explanations reach the bottom level (asserted facts). Figure 7.7 shows the sequence, while blue arrows point to the domain explanations, green arrows point to the original rules, black arrows point to the asserted facts and red arrows point to the explanations with other rules.

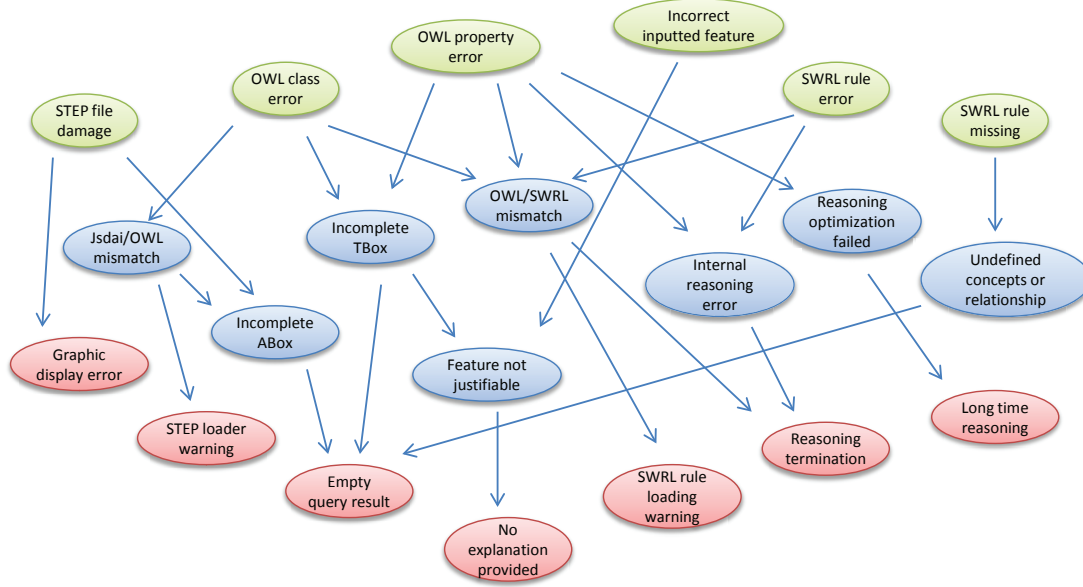


Figure 7.8: Probabilistic causality network for software diagnosis in OAES

## 7.5 Software Diagnostic Model

The software diagnostic model generates explanations for user observations of the system's unexpected behaviors, where AR is the most common cognitive model used in this case. The diagnostic model implements MRICN to provide an interactive AR function in generating explanations for the observations. MRICN interacts with users and draws new information to allow for reflective searching for the optimal sets of knowledge with the maximal probability gain. Details of MRICN have been discussed in Chapter 5. The original objective of this diagnostic model is to diagnose other AFR software products as one of the major functions of OAES. However, since most of existing AFR systems are theoretical models and commercial AFR software products are rarely available, the diagnostic model in this study diagnoses the software of OAES itself, which gives a demonstration of how MRICN can be applied into software



Table 7.3: Probabilistic causalities with conditional probabilities

Cause(C)	Effect(E)	P(C/E)	P( $\bar{C}$ /E)	P(C/ $\bar{E}$ )	P( $\bar{C}$ / $\bar{E}$ )
STEP file damage	Graphic display error	0.95	0.05	0.2	0.8
STEP file damage	Incomplete ABox	0.4	0.6	0.1	0.9
OWL class error	Jsdai/OWL mismatch	0.7	0.3	0.35	0.75
OWL class error	Incomplete TBox	0.8	0.2	0.3	7
OWL class error	OWL/SWRL mismatch	0.7	0.3	0.25	0.75
OWL property error	Incomplete TBox	0.75	0.25	0.2	0.8
OWL property error	OWL/SWRL mismatch	0.55	0.45	0.1	0.9
OWL property error	Internal reasoning error	0.65	0.35	0.1	0.9
OWL property error	Reasoning optimization failed	0.9	0.1	0.5	0.5
Incorrect inputted feature	Feature not justifiable	0.8	0.2	0.05	0.95
SWRL rule error	OWL/SWRL mismatch	0.6	0.4	0.3	0.7
SWRL rule error	Internal reasoning error	0.8	0.2	0.2	0.8
SWRL rule missing	Undefined concepts or relationships	0.95	0.05	0.25	0.75
Jsdai/OWL mismatch	STEP loader warning	0.95	0.05	0.15	0.85
Jsdai/OWL mismatch	Incomplete ABox	0.6	0.4	0.1	0.9
Incomplete ABox	Empty query result	0.4	0.6	0.1	0.9
Incomplete TBox	Empty query result	0.65	0.35	0.05	0.95
Incomplete TBox	Feature not justifiable	0.6	0.4	0.3	0.7
Feature not justifiable	No explanation provided	0.9	0.1	0.1	0.9
OWL/SWRL mismatch	SWRL rule loading warning	0.95	0.05	0.05	0.95
OWL/SWRL mismatch	Reasoning termination	0.6	0.4	0.35	0.65
Internal reasoning error	Reasoning termination	0.8	0.2	0.15	0.85
Reasoning optimization failed	Long time reasoning	0.6	0.4	0.15	0.95

diagnostics. Figure 7.8 shows the network. The network can be easily extended for other applications by filling different propositions.

In Figure 7.8, the red nodes represent the system faults that may be observed by users, and the green nodes represent hypotheses that are potential causes for the system faults. The blue nodes are the intermediate propositions linking causes to their effects. Table 7.3 shows the conditional probabilities between the nodes. For each proposition, there exists a corresponding OWL class, which shares the same name with the node. Descriptions of the propositions are also captured using “rdfs:comment” property, and are also presented upon the user’s request by clicking the corresponding

HTML links.

In the software diagnostic model, MRICN is constructed with conditional probabilities listed in Table 7.3, and is used to perform an interactive diagnostic process. For example, a user may have observed that the system failed to provide explanations for the input feature. The following dialog is then carried out as shown in Figure 7.9.

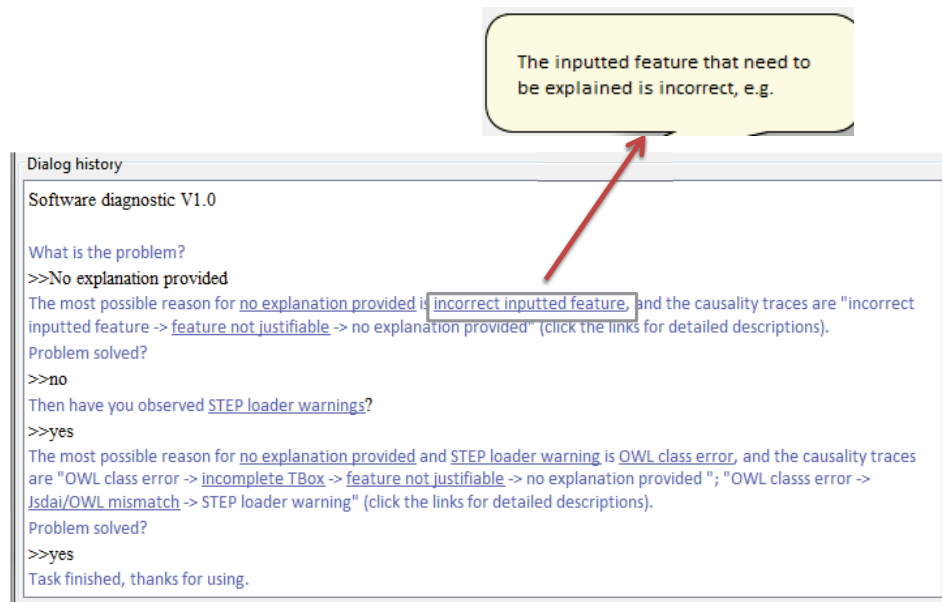


Figure 7.9: The dialog history in the software diagnostic model

In Figure 7.9, the blue texts are produced by the system and the black texts come from the user. The domain explanations for the propositions can be provided by clicking the corresponding HTML links, as the red arrow shows.

## 7.6 Usage of OAES

This section briefly discusses the usage of OAES by introducing the user interface of the software.

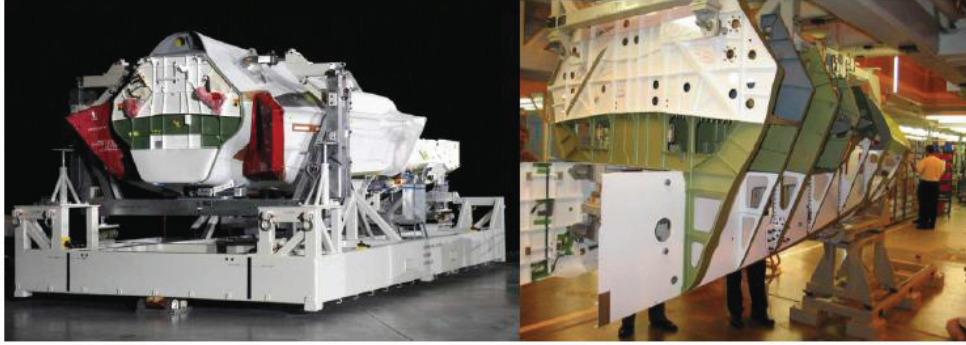


Figure 7.10: Internal structure of the F-35 strike fighter

After the ontology (“.owl” file) and the STEP file are loaded into the memory, the main interface of the software comes out as shown in Figure 7.11. The test file is a simplified model of the internal structure of a modern fighter aircraft shown in Figure 7.10. The top left panel in the main interface is the Explanation & Query panel carrying out the explanation and query tasks. The ontology panel in the middle left displays the rules, classes and properties in the ontology. The bottom left panel is the console, which displays the system’s running state, or warnings and errors that have been produced. The visualization of the input part is displayed in the right of the interface, where JAVA3D technique is used.

The ontology panel includes three tabs: ontology class tab, ontology property tab and rule tab. The ontology class tab (shown in Figure 7.12) shows the class hierarchy within the ontology, as well as some basic information, including name,

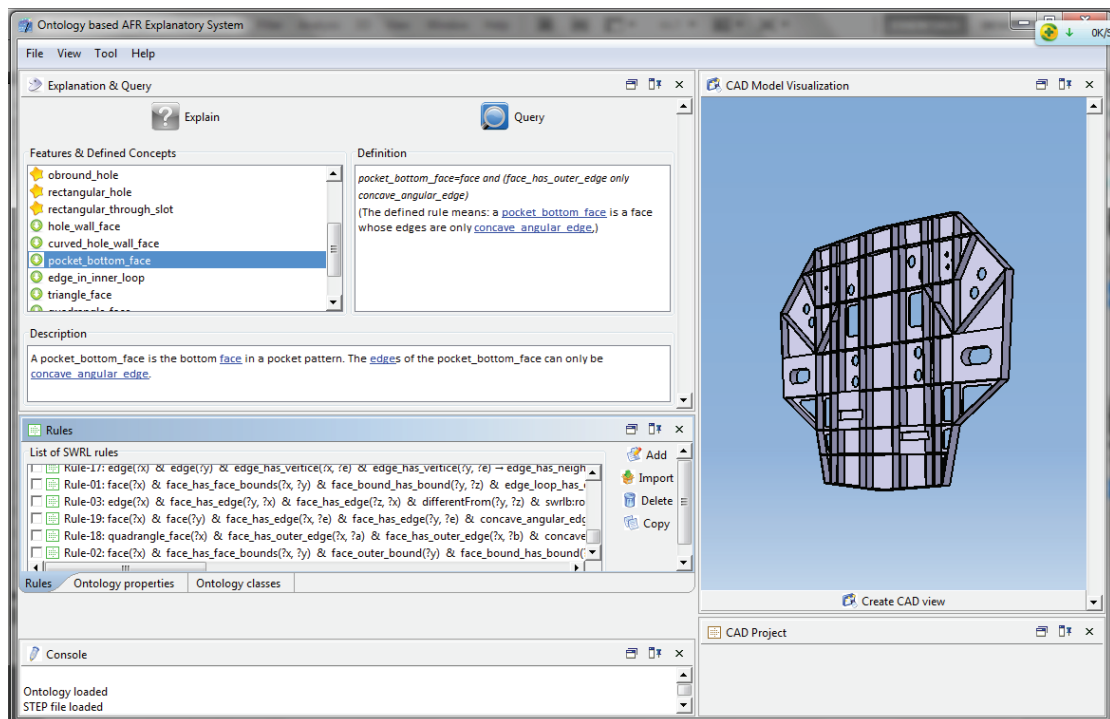
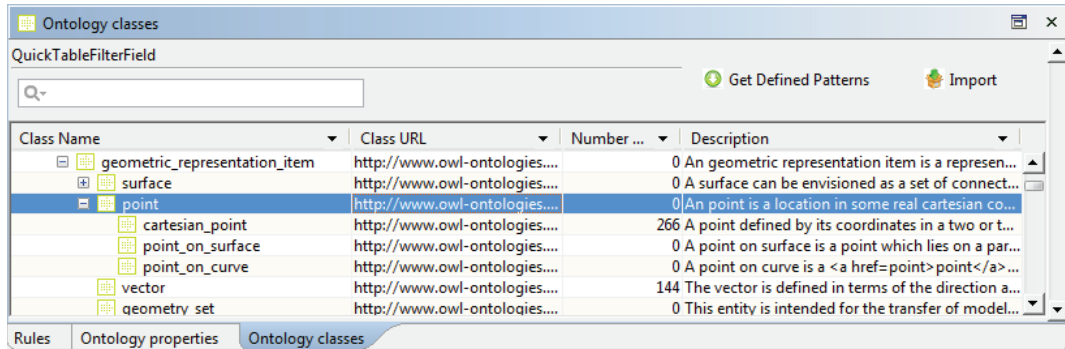


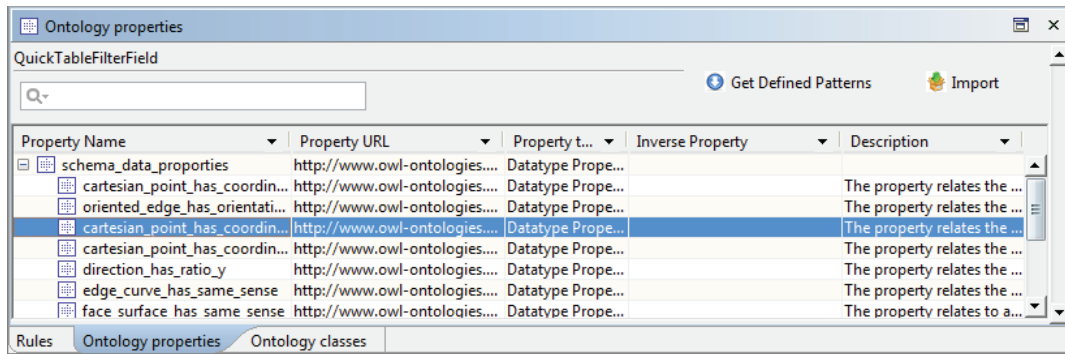
Figure 7.11: The main interface of OAES

URL, number of instances and short description. The property tab, as shown in Figure 7.13, is similar to the class tab, and the rule tab is shown in Figure 7.14.



Class Name	Class URL	Number	Description
geometric_representation_item	http://www.owl-ontologies...	0	An geometric representation item is a represen...
surface	http://www.owl-ontologies...	0	A surface can be envisioned as a set of connect...
point	http://www.owl-ontologies...	0	An point is a location in some real cartesian co...
cartesian_point	http://www.owl-ontologies...	266	A point defined by its coordinates in a two or t...
point_on_surface	http://www.owl-ontologies...	0	A point on surface is a point which lies on a par...
point_on_curve	http://www.owl-ontologies...	0	A point on curve is a <a href=point>point</a>...
vector	http://www.owl-ontologies...	144	The vector is defined in terms of the direction a...
geometry set	http://www.owl-ontologies...	0	This entity is intended for the transfer of model...

Figure 7.12: Ontology class tab



Property Name	Property URL	Property type	Inverse Property	Description
schema_data_properties	http://www.owl-ontologies...	Datatype Prope...		
cartesian_point_has_coordin...	http://www.owl-ontologies...	Datatype Prope...		The property relates the ...
oriented_edge_has_orientati...	http://www.owl-ontologies...	Datatype Prope...		The property relates the ...
cartesian_point_has_coordin...	http://www.owl-ontologies...	Datatype Prope...		The property relates the ...
cartesian_point_has_coordin...	http://www.owl-ontologies...	Datatype Prope...		The property relates the ...
direction_has_ratio_y	http://www.owl-ontologies...	Datatype Prope...		The property relates the ...
edge_curve_has_same_sense	http://www.owl-ontologies...	Datatype Prope...		The property relates the ...
face surface has same sense	http://www.owl-ontologies...	Datatype Prope...		The property relates to a...

Figure 7.13: Ontology property tab

Once the user double clicks a class in the class tab, details of the class are shown in Figure 7.15. In the class detail panel, the super-classes (top left), the sub-classes (top middle) and the instances (top right) are displayed on the top, and the description of the class is shown on the bottom.

In the Explanation & Query panel, shown in Figure 7.16, features and defined

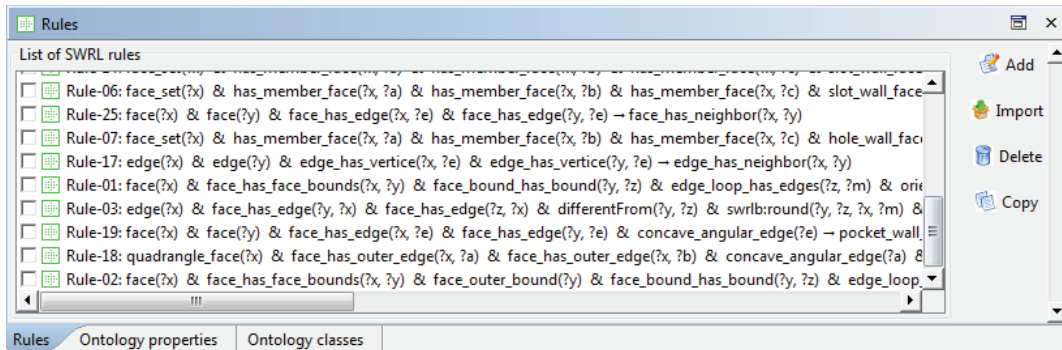


Figure 7.14: The SWRL rule tab

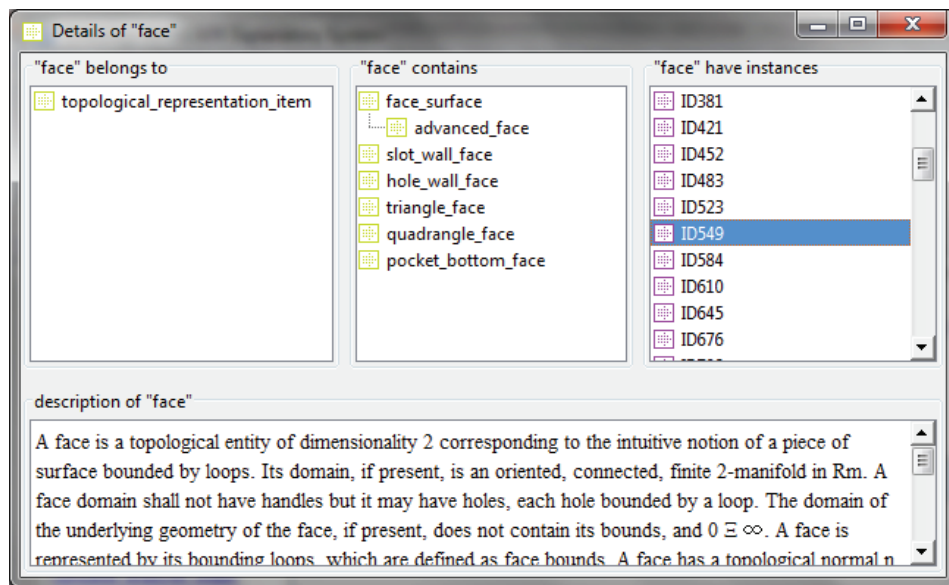


Figure 7.15: Class detail panel for *face\_bound*

concepts are listed on the left. Once the user has selected an item in the list, the definition and the description of the selected concept are displayed on the right and bottom respectively. Two buttons sit on the top of the panel, representing two reasoning tasks: the left one is used for generating explanation for a *face\_set* which has been recognized as the selected feature, and the right one is used for querying instances for the selected feature.

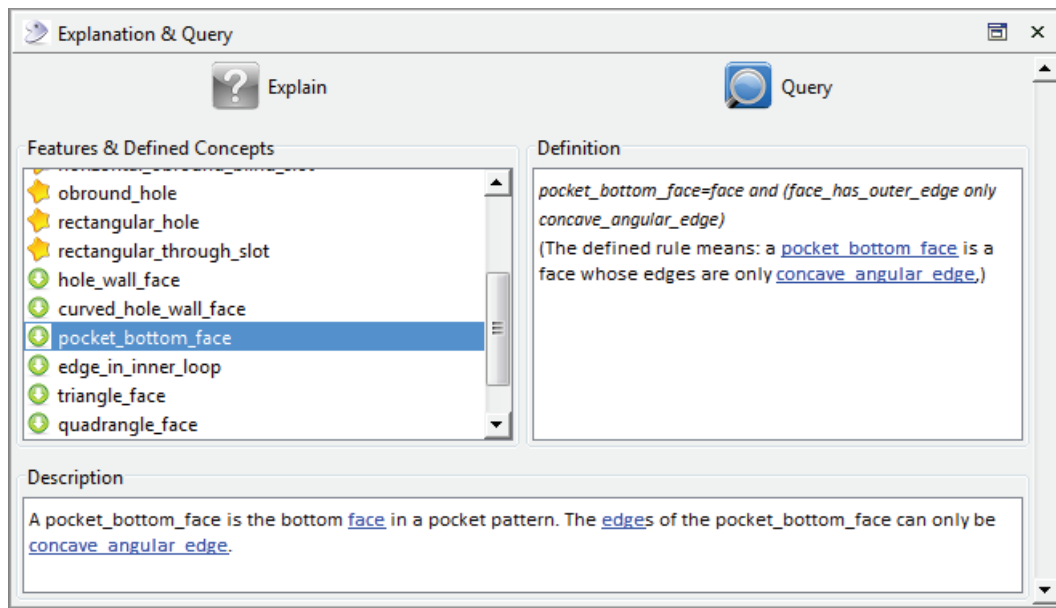


Figure 7.16: Explanation & query panel

The explanation here refers to the decision explanation, namely the explanation of why a *face\_set* has been recognized as a particular feature. For example, if a *face\_set* has been recognized as an *obround\_hole* and the user wants to know why, he or she can select the *obround\_hole* from the feature list in the reasoning panel, and then click the “explanation” button. The explanation panel will then appear, as shown in

Figure 7.17. In the explanation panel, the description of the selected feature is shown on the top right. The member faces of the *face\_set* that has been recognized as an *obround\_hole* can be inputted at the middle right of the panel. After the *face\_set* has been inputted, the user needs to press the button on the right asking for the explanation. The visualization of the model with inputted faces highlighted is then shown on the left, and the explanation is displayed on the bottom right.

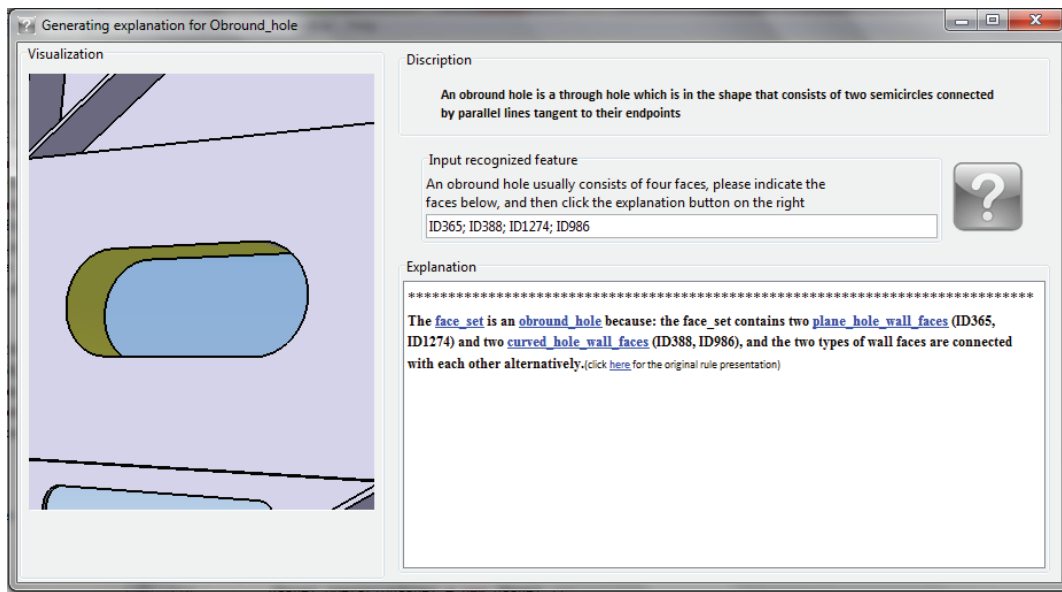


Figure 7.17: The explanation panel for *obround\_hole* feature

In this case, the system firstly explains why the *face\_set* is an *obround\_hole* using the primary recognition rule (as shown in Figure 7.17). The explanation, as Figure 7.18 shows, can then be extended if the user requires further information (the original SWRL rules are shown in the balloons, and further explanations are displayed subsequently in the explanation display area). For example, if the user wants to know why “ID388” is recognized as a *curved\_hole\_wall\_face*, he or she can click the correspond-



ing HTML link *curved\_hole\_wall\_face*(ID388) in the first balloon. The explanation is then extended, as shown in the second part of the explanation. The third part is generated once the user want to know why ID388 is a *hole\_wall\_face*.

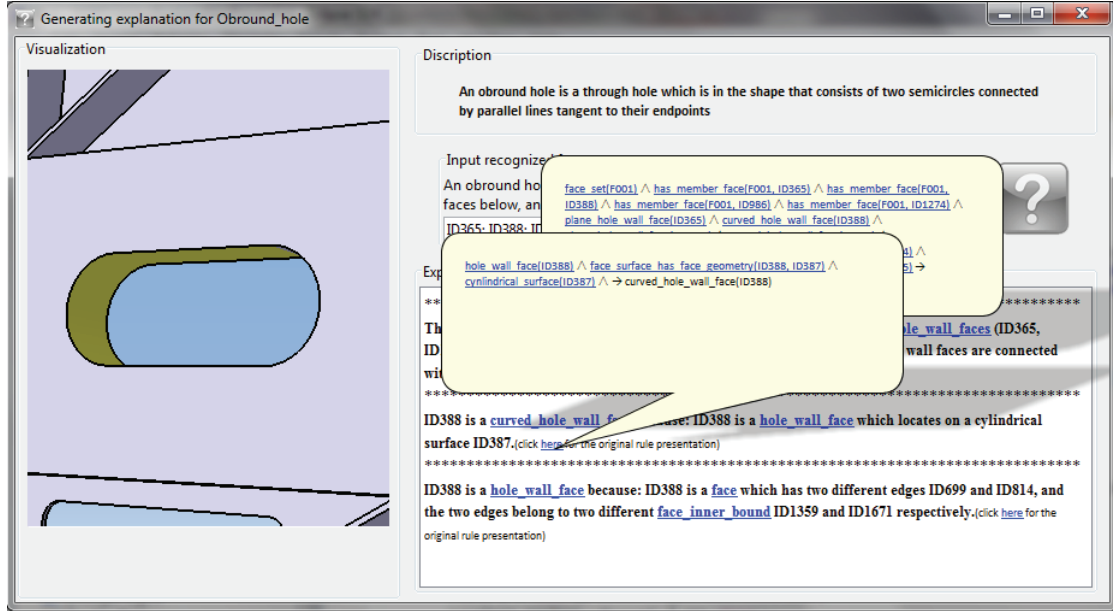


Figure 7.18: The explanation panel for *obround\_hole* feature cont'd

Another two example cases of generating explanation are shown in Figure 7.19 and Figure 7.20: one is for explaining a *triangular\_pocket*, and the other is for a *regular\_through\_slot*.

In addition, if the user presses the “query” button in the Explanation & Query panel with the class *concave\_angular\_edge* selected, a query panel comes out as Figure 7.21 shows. In the query panel, the user can select query tasks on the top left, and the query result is displayed on the top right. The user can also click each instance in the query result to get an explanation about why the instance belongs to the class,

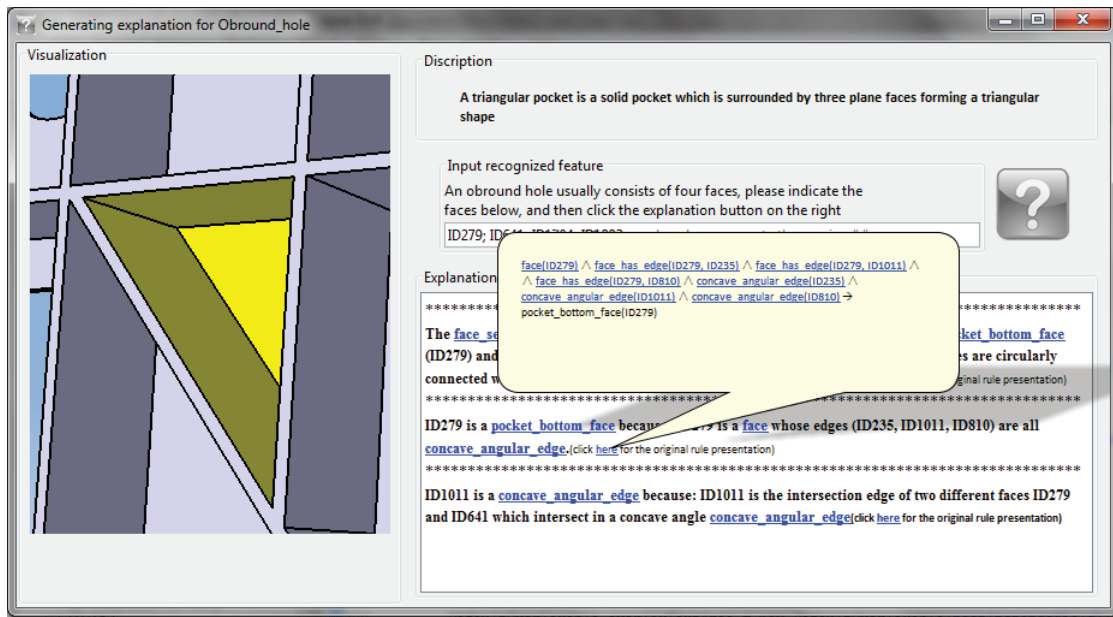


Figure 7.19: The explanation panel for *triangular\_pocket* feature

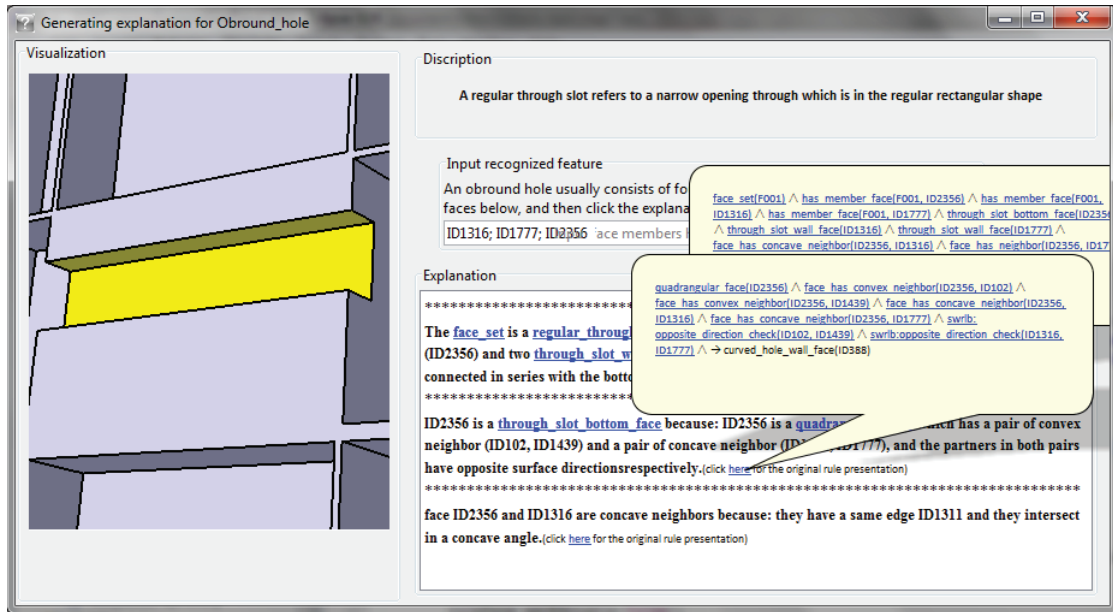


Figure 7.20: The explanation panel for *regular\_through\_slot* feature

which is shown in the bottom of the panel.

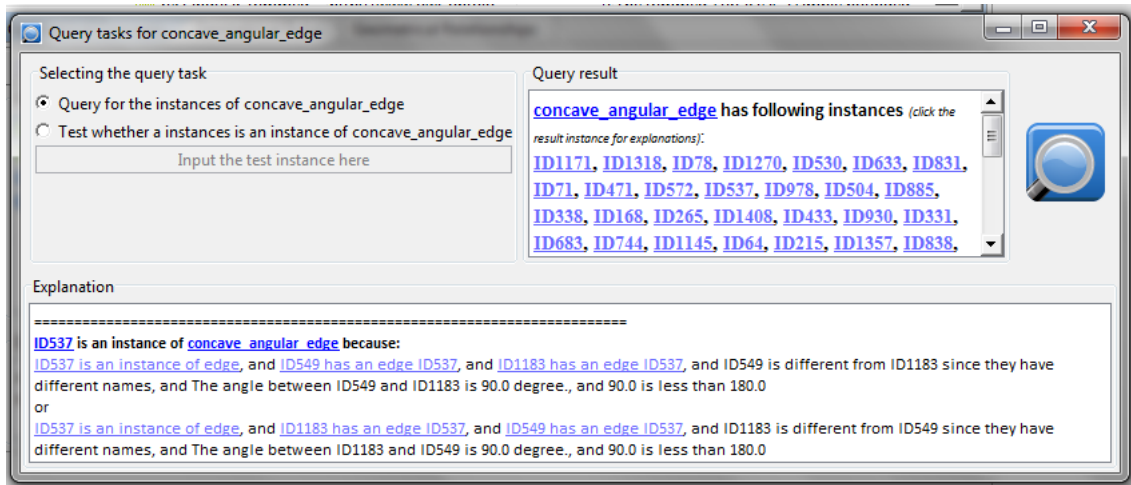


Figure 7.21: The query panel for *concave\_angular\_edge*

Finally, the software diagnostic model can be started from the tool menu on the top of the main interface, and the interface of the model is shown in Figure 7.22. Taking the example shown in Figure 7.9, the system firstly asks the user to tell what is the problem by selecting one of the prompt options, as shown in the balloon in Figure 7.22. After the user tells that the problem is “no explanation provided”, the system, as Figure 7.23 shows, then replies that the most possible reason for “no explanation provided” is “incorrect inputted feature”. The dialog history is recorded on the bottom of the panel. In addition, descriptions (domain explanation) of the propositions can be accessed by clicking the corresponding HTML link (shown in the smaller balloon in Figure 7.23).

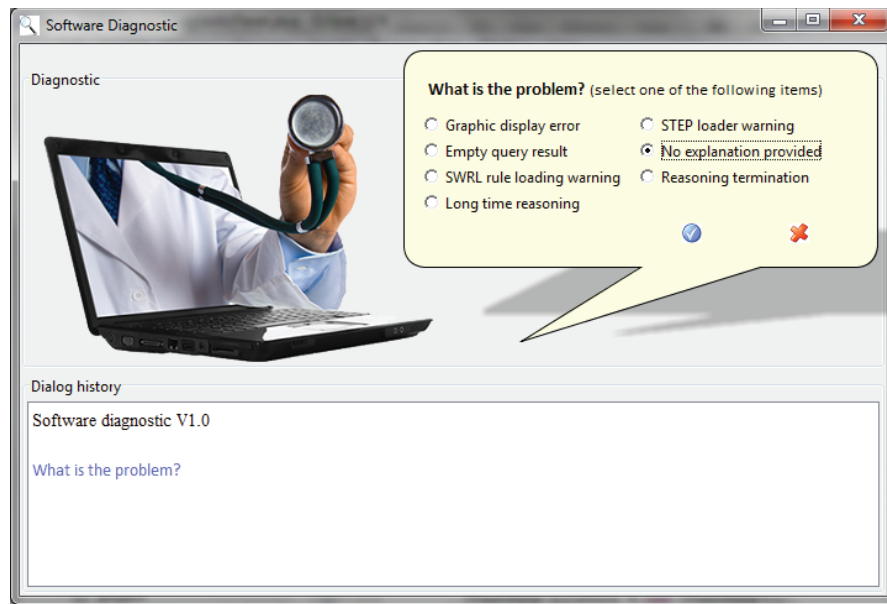


Figure 7.22: Interface of the software diagnostic model

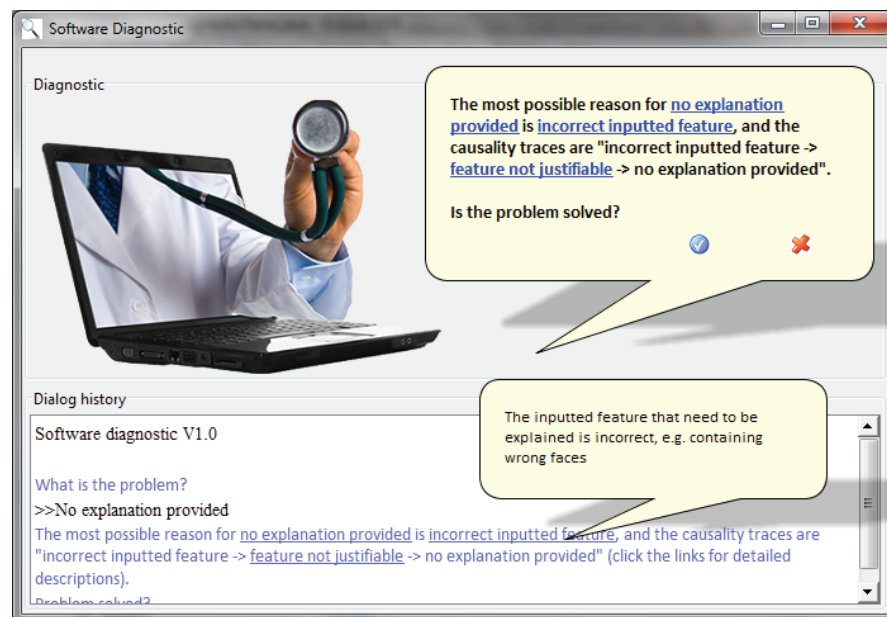


Figure 7.23: Interface of the software diagnostic model cont'd

## 7.7 Summary

In this chapter, the explanation framework has been implemented to generate explanations for AFR systems. A novel AFR explanatory system, OAES, has been thus developed. OAES focuses on STEP-based AFR systems, and owns an independent explanatory OWL ontology, which contains all the information needed for generating explanations.

All of the three types of explanation have been integrated in OAES. For decision explanation, OAES generates explanations for a recognized feature by giving a clear and understandable description of the topology of the face set, without referring to the actual decision making process in the AFR systems. For domain explanations, the information is captured in the ontology using RDF utility properties, and is presented to the user upon request. For software diagnostic, MRICN is implemented to diagnose system failures or software running problems using the probabilistic causal network presented in Section 7.5. The OAES has been implemented using JAVA, and the interface has been discussed.

In fact, OAES is simply an application case study used to demonstrate the explanation framework. Other applications can also be considered. For example, a STEP-based mechanical assembly planning system was proposed in [153]. The system represents the product assembly model with STEP, using mainly the entities of integrated resources and partially the self-defined entities, and is able to plan assembly sequences using a knowledge-based planning approach, which generates all feasible assembly sequences of the product through reasoning and decomposition process. The explanation model for this system will be similar to OAES, except it contains different

OWL classes and properties for the self-defined entities, and different rules that are summarized from the knowledge used for assembly sequence planning. Other applications, such as a STEP-based process planning System [4] and a product configuration system , all have the potential for explanation using the framework proposed in this study.

# Chapter 8

## Conclusion and Future Studies

### 8.1 Introduction

This chapter concludes the whole work proposed in the thesis, and discusses the current limitations and corresponding potential future works.

The rest of this chapter is organized as follows: Section 8.2 concludes the thesis; Section 8.3 presents potential future works; Section 8.4 summarizes the chapter.

### 8.2 Conclusions

The thesis has proposed an innovative intelligent explanation framework for KBSs. Different with existing models which are usually built inside a KBS and generate explanations based on the actual decision making process, in our framework, the explanation model stands outside the KBS and attempts to reconstruct the explanations for the KBS behaviors by “concocting” as a third party. The knowledge structure and reasoning methods have thus been optimized particularly for explanation generation

within this independent explanation model, so that the quality of explanation has been improved.

In this thesis, introduction and literature review have been given in Chapter 1 and Chapter 2. Chapter 3 has given an overview of the framework. Three sub-models, EKB, DEM and SDM have been included. Technical details of a novel reasoner BCAR that is used in DEM, and a novel probabilistic network MRICN that is used in SDM, have been discussed in Chapter 4 and Chapter 5 respectively. The explanation framework has been demonstrated using a concrete AFR scenario. Backgrounds of AFR have been introduced in Chapter 6, and a novel AFR explanatory system OAES has been proposed in Chapter 7.

The major results and contributions of the thesis are summarized as below:

1. The explanation framework has been designed to cover all the three types of explanation: 1) decision explanation, which helps users understand how a KBS reached its conclusions; 2) domain explanation, which provides detailed descriptions of the concepts and relationships within the domain; 3) software diagnostic, which diagnoses user observations of unexpected behaviors of the system or some relevant domain phenomena.
2. The explanation framework contains an independent explanation model that stands outside the KBS, and generates explanations by producing a justification that does not relate to how the decision was actually made. The framework generally includes three sub-models: 1) EKB, which refers to a separate explanative knowledge base that is constructed as an ontology using OWL and SWRL; 2) DEM, which is used to provide decision explanations, where reasoning issues



have been highlighted; SDM, which is used to provide software diagnostics, where the core activity is AR.

3. OWL and SWRL have been introduced to construct the explanatory ontology for EKB. The ontology is created in Protege ontology editor, and is saved as a “.owl” file. Methods for mapping domain knowledge to the ontology have been discussed in Chapter 3, which can be generally divided into four parts: 1) using ontology TBox to capture domain concepts and relationships; 2) using RDF utility properties to capture descriptive information of the domain concepts and relationships as annotations of relevant classes and properties; 3) using SWRL rules or OWL ECAs to capture design rules and constraints; 4) using ontology ABox to capture a particular input instance. Details of the above process have been discussed in Chapter 7 with a concrete case.
4. BCAR has been developed to efficiently reason through the ontology to generate explanations. The reasoner mainly focuses on solving the assumption mismatch problem between OWL ontology and CWA-based information model, and has achieved good performance in generating explanations by using backward chaining mechanism, as explanation generation is basically a goal-oriented task.
5. MRICN has been developed for SDM. The network draws ideas from the actual human-human medical diagnostic process, and performs interactive AR using probabilistic computational methods. OP theory has been integrated into the network enabling MRICN to “guess” a plausible explanation, despite the absence of detailed joint probabilistic information. It has been shown that MRICN is able to interact with users and seek unobserved evidence in order to maximize

the fitness of its knowledge structures.

6. The explanation framework has been implemented and demonstrated within a concrete AFR scenario as OAES, which generates explanations for any AFR system that takes STEP files (AP203) as inputs. In this case, three types of explanations are provided as: 1) justification of how a particular feature is recognized; 2) descriptive information for geometric concepts and relationships regarding B-rep of solid models; 3) reasons for software unexpected behaviors while using OAES. In OAES, the “.owl” file, which contains only TBox information, is first inputted to the system, and is then instantiated by loading a STEP file. The user can then input the feature that has been recognized for explanation. The explanation is presented as HTML text, which allows the user to get descriptions of concepts or relationships mentioned by clicking the corresponding HTML links. Once the user finds system errors or unexpected system behaviors, the software diagnostic panel can be used to assist the user to find out the reason for the observations.

### 8.3 Limitations and Future Works

Finally, the current limitations of the explanation framework and corresponding future works are listed below.

- DEM only involves simple interactions allowing the user to ask for further detailed explanations. Complex techniques regarding user profile modeling and context handling are not integrated, so that adaptive explanations are not avail-

able in the current stage. In fact, capturing user profile in an ontology is not difficult. Some works have already been proposed in this area [38] [115]. Once the user profile is captured, providing adaptive explanations is also possible, e.g. encode two rule bases or concept descriptions with different depths, and present them to the user according to the user’s expertise. A fresh reasoning method, which generates explanations taking account the user profile, may be needed. Applying this change to the model will be relatively simple due to the reconstructive explanation method. It offers great flexibility in designing the system, as we can plan everything we want without considering the actual decision making process in the KBS.

- BCAR needs to be improved in the following aspects:
  - support more OWL constructs: Some of the OWL constructs are not supported in BCAR, and will result in reasoning errors if they are contained in the input ontology, such as transitive and symmetric property;
  - support more data formats: BCAR can only process “Float”, “Integer”, “Boolean” and “String” as basic data types. Other data types, such as date and time which are supported by both OWL and STEP, are not available in BCAR;
  - enrich built-in functions: The current SWRL built-in function library in BCAR is relatively small. Only few built-in functions can be processed. Future works in building more built-in functions can greatly improve the computational capability of BCAR.

- Although DEM and SDM are both integrated in the explanation framework, they are not quite closely related to each other. They are more like two independent systems where the only relation between them is that they share the same explanatory ontology for providing domain explanations. Furthermore, neither the probabilistic causal relationships in MRICN can be captured in the OWL ontology, nor BCAR can be applied to reason through the probabilistic network. Future research addressing these issues, thus, has two stages:
  - The first stage is to study how to implement MRICN using OWL constructs or SWRL rules, which refers to the research regarding how to put uncertainty into OWL and SWRL. For OWL, existing related works includes Probabilistic Description Logics [79] and Bayesian approach based OWL ontology [29]. For SWRL, a proposal of Fuzzy Extension of SWRL was given in [119], but the probabilistic extension has never been discussed. Future work will be along these directions.
  - Once MRICN is implemented using OWL and SWRL, there is a need to develop an ontology reasoner to reason through the ontology under uncertainty. Very few studies have been proved to be effective and efficient. Major difficulties include maintaining ontology consistence, dealing with logic quantifiers, and etc.
- OAES contains a relatively small rule base, so that only basic features can be justified and explained. Complex features, especially intersecting features, are not available in the current stage. To improve the performance of OAES, it is important to define more rules allowing for more complex features to be

explained, which is also the key motivation for many existing AFR systems.

## 8.4 Summary

This chapter has finally concluded the thesis and has discussed several limitations of the framework and corresponding future research directions. In general, the thesis has proposed an explanation framework, including theoretical foundation and computational algorithms, and has demonstrated the framework through its application to AFR systems. The results have shown that the general goal of this study, building an computational framework for explanation generation in KBSs, has been achieved.

# Bibliography

- [1] P. Achinstein. *The Nature of Explanation*. Oxford University Press, 1985.
- [2] P. Achinstein. *Evidence, explanation, and realism: essays in the philosophy of science*. Oxford University Press, 2010.
- [3] R. Akerkar and P. Sajja. *Knowledge-Based Systems*. Jones and Bartlett, 2010.
- [4] S. Amaitik and S. Kilic. An intelligent process planning system for prismatic parts using STEP features. *The International Journal of Advanced Manufacturing Technology*, 31:978–993, 2007.
- [5] O. Andersen and G. Vasilakis. Building an ontology of CAD model information. In *Geometric Modelling, Numerical Simulation, and Optimization*, pages 11–40. 2007.
- [6] C. Angeli. Diagnostic expert systems: from expert’s knowledge to real-time systems. *Advanced Knowledge Based Systems: Model, Application and Research*, 1:50–73, 2010.
- [7] G. Antoniou and F. Harmelen. *A semantic Web primer*. MIT Press, 2004.

- [8] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
- [9] B. R. Babic, N. Nesic, and Z. Miljkovic. A review of automated feature recognition with rule-based pattern recognition. *Computers in Industry*, 59:321–337, 2008.
- [10] J. Balic. Intelligent CAD/CAM systems for CNC programming - an overview. *Advances in Production Engineering & Management*, 1:13–22, 2006.
- [11] M. B. Bassat, R. W. Carlson, V. K. Puri, M. D. Davenport, M. D. Schriver, M. Latif, R. Smith, L. D. Portigal, E. H. Lipnick, and M. H. Weil. Pattern-based interactive diagnosis of multiple disorders: the MEDAS system. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2:148–160, 1980.
- [12] R. D. Beer. A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence*, 72:173–215, 1995.
- [13] R. D. Beer. The dynamics of adaptive behavior: a research program. *Robotics and Autonomous Systems*, 20:257–289, 1997.
- [14] D. Benyon and D. Murray. Applying user modeling to human-computer interaction design. *Artificial Intelligence Review*, 7:199–225, 1993.
- [15] M. P. Bhandarkar and R. Nagi. STEP-based feature extraction from STEP geometry for agile manufacturing. *Computers in Industry*, 41:3–24, 2000.

- [16] J. Bock, P. Haase, Q. Ji, and R. Volz. Benchmarking OWL reasoners. In *Proceedings of the ARea2008 Workshop*, 2008.
- [17] Z. Bofeng, W. Na, W. Gengfeng, and L. Sheng. Research on a personalized expert system explanation method based on fuzzy user model. In *Proceedings of the Fifth World Congress on Intelligent Control and Automation*, pages 3996–4000, 2004.
- [18] T. Bylander and B. Chandrasekaran. Generic tasks for knowledge-based reasoning: the right level of abstraction for knowledge acquisition. *International Journal of Man-Machine Studies*, 26:231–243, 1987.
- [19] S. Carberry. Modeling the user’s plans and goals. *Computational Linguistics*, 14:23–37, 1988.
- [20] A. Castro and V. Miranda. Knowledge discovery in neural networks with application to transformer failure diagnosis. *IEEE Transactions on Power Systems*, 20:717–724, 2005.
- [21] A. Cawsey. *Explanation and interaction: the computer generation of explanatory dialogues*. MIT Press, 1992.
- [22] A. Cawsey. Developing an explanation component for a knowledge-based system: discussion. *Expert Systems with Applications*, 8:527–531, 1995.
- [23] B. Chandrasekaran and S. Mittal. Deep versus compiled knowledge approaches to diagnostic problem-solving. *International Journal of Man-Machine Studies*, 19:425–436, 1983.



- [24] B. Chandrasekaran, M. C. Tanner, and J. R. Josephson. Explaining control strategies in problem solving. *IEEE Expert: Intelligent Systems and Their Applications*, 4:9–24, 1989.
- [25] E. Charniak and S. E. Shimony. Cost-based abduction and MAP explanation. *Artificial Intelligence*, 66:345–374, 1994.
- [26] W. J. Clancey and R. Letsinger. NEOMYCIN: reconfiguring a rule-based expert system for application to teaching. Technical report, Stanford, CA, USA, 1982.
- [27] L. M. de Campos, J. A. Gamez, and S. Moral. Simplifying explanations in bayesian belief networks. *International Journal of Uncertainty, Fuzziness and KnowledgeBased Systems*, 2:167–196, 1994.
- [28] L. Ding and Y. Yue. Novel ANN-based feature recognition incorporating design by features. *Computers in Industry*, 55:192–222, 2004.
- [29] Z. Ding, Y. Peng, and R. Pan. A bayesian approach to uncertainty modeling in OWL ontology. In *Proceedings of the International Conference on Advances in Intelligent Systems - Theory and Applications*, pages 9–13, 2004.
- [30] B. Falciديو and F. Giannini. Automatic recognition and representation of shape-based features in a geometric modeling system. *Computer Vision, Graphics, and Image Processing*, 48:93–123, 1989.
- [31] S. Fei and X. Zhang. Fault diagnosis of power transformer based on support vector machine with genetic algorithm. *Expert Systems with Applications*, 36(8):11352 – 11357, 2009.

- [32] A. Fiedler. P.REX: an interactive proof explainer. In *Proceedings of the First International Joint Conference on Automated Reasoning*, pages 416–420, 2001.
- [33] T. Finin and G. Morris. Abductive reasoning in multiple fault diagnosis. *Artificial Intelligence Review*, 3:129–158, 1989.
- [34] B. Fraassen. *The scientific image*. Clarendon Press, 1980.
- [35] C. Genest and J. V. Zidek. Combining probability distributions: a critique and an annotated bibliography. *Statistical Science*, 1:114–135, 1986.
- [36] S. A. Ghafour, P. Ghodous, B. Shariat, and E. Perna. Towards an intelligent CAD models sharing based on semantic web technologies. In *Proceedings of the 15th ISPE International Conference on Concurrent Engineering*, pages 195–203, 2008.
- [37] A. Glass, D. L. McGuinness, and M. Wolverton. Toward establishing trust in adaptive agents. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 227–236, 2008.
- [38] M. Golemati, A. Katifori, C. Vassilakis, G. Lepouras, and C. Halatsis. Creating an ontology for the user profile: Method and applications. In *Proceedings of the First International Conference on Research Challenges in Information Science*, pages 407–412, 2007.
- [39] S. Gregor and I. Benbasat. Explanations from intelligent systems: theoretical foundations and implications for practice. *MIS Quarterly*, 23:497–530, 1999.

- [40] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3:158–182, 2005.
- [41] V. Haarslev, K. Hidde, R. Möller, and M. Wessel. The RacerPro knowledge representation and reasoning system. <http://www.semantic-web-journal.net>, 2011.
- [42] J. Han, M. Kang, and H. Choi. STEP-based feature recognition for manufacturing cost optimization. *Computer-Aided Design*, 33:671–686, 2001.
- [43] J. Han, M. Pratt, and W. C. Regli. Manufacturing feature recognition from solid models: a status report. *IEEE Transactions on Robotics and Automation*, 16:782–796, 2000.
- [44] J. Han and A. A. Requicha. Integration of feature based design and feature recognition. *Computer-Aided Design*, 29:393–403, 1997.
- [45] M. Harbers, K. van den Bosch, and J. J. Meyer. Design and evaluation of explainable BDI agents. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 125–132, 2010.
- [46] D. W. Hasling, W. J. Clancey, and G. Rennels. Strategic explanations for a diagnostic consultation system. *International Journal of Man-Machine Studies*, 20:3–19, 1984.
- [47] S. R. Haynes. *Explanation in Information Systems: A Design Rationale Approach*. PhD thesis, London School of Economics and Political Science, 2001.

- [48] S. R. Haynes, M. A. Cohen, and F. E. Ritter. Designs for explaining intelligent agents. *International Journal of Human-Computer Studies*, 67:90–110, 2009.
- [49] D. Heckerman, J. S. Breese, and K. Rommelse. Decision-theoretic troubleshooting. *Communications of the ACM*, 38:49–57, 1995.
- [50] C. G. Hempel and P. Oppenheim. Studies in the logic of explanation. *Philosophy of Science*, 15:135–175, 1948.
- [51] J. L. Herlocker, J. A. Konstan, and J. Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 241–250. ACM, 2000.
- [52] J. Herrmann, M. Kloth, and F. Feldkamp. The role of explanations in an intelligent assistant system. *Artificial Intelligence in Engineering*, 12:107 – 126, 1998.
- [53] Hewlett-Packard Development Company. JENA - a semantic web framework for java. <http://jena.sourceforge.net/index.html>, 2002.
- [54] G. E. Hinton. Products of experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks*, pages 1–6, 1999.
- [55] J. R. Hobbs, M. E. Stickel, D. E. Appelt, and P. Martin. Interpretation as abduction. *Artificial Intelligence*, 63:69 – 142, 1993.
- [56] M. Horridge, S. Jupp, G. Moulton, A. Rector, R. Stevens, and C. Wroe. A practical guide to building OWL ontologies using protege 4 and co-ode tools edition

- 1.1. <http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/>, 2008.
- [57] I. Horrocks and P. F. Patel-Schneider. A proposal for an OWL rules language. In *Proceedings of the Thirteenth International World Wide Web Conference*, pages 723–731, 2004.
- [58] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosz, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C member submission, 2004.
- [59] U. Hustadt. Reducing SHIQ- description logic to disjunctive datalog programs. In *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 152–162, 2004.
- [60] M. Jang and J. chan Sohn. Bossam: An extended rule engine for OWL. *Rules and Rule Markup Languages for the Semantic Web*, 3323/2004:128–138, 2004.
- [61] W. L. Johnson. Agents that learn to explain themselves. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1257–1263, 1994.
- [62] T. J. Jones, C. A. Reidsema, and A. Smith. Automated feature recognition system for supporting conceptual engineering design. *KES Journal*, 10:477–492, 2006.
- [63] J. R. Josephson and S. G. Josephson. *Abductive Inference*. Cambridge University Press, 1994.

- [64] S. Joshi and T. C. Chang. Graph-based heuristics for recognition of machined features from a 3D solid model. *Computer-Aided Design*, 20:58–66, 1988.
- [65] J. Kalagnanam and M. Henrion. A comparison of decision alaysis and expert rules for sequential diagnosis. In *Proceedings of the Fourth Annual Conference on Uncertainty in Artificial Intelligence*, pages 271–282, 1990.
- [66] T. R. Kannan and M. S. Shunmugam. Processing of 3D sheet metal components in STEP AP-203 format. Part I: feature recognition system. *International Journal of Production Research*, 47:941–964, 2009.
- [67] R. Kass. Building a user model implicitly from a cooperative advisory dialog. *User Modeling and User-Adapted Interaction*, 1:203–258, 1991.
- [68] J. Kay. Reusable tools for user modelling. *Artificial Intelligence Review*, 7:241–251, 1993.
- [69] F. Keil and R. Wilson. *Explanation and cognition*. Bradford Books. MIT Press, 2000.
- [70] F. C. Keil. Explanation and understanding. *Annual Review of Psychology*, 57:227–254, 2006.
- [71] K. Y. Kim, D. G. Manley, and H. Yang. Ontology-based assembly design and information sharing for collaborative product development. *Computer-Aided Design*, 38:1233–1250, 2006.
- [72] Y. Kim. Recognition of form features using convex decomposition. *Computer-Aided Design*, 24:461 – 476, 1992.

- [73] P. Kitcher and W. Salmon. Van Fraassen on explanation. *The Journal of Philosophy*, 84:315–330, 1987.
- [74] S. I. Krima, R. Barbau, X. Fiorentini, R. Sudarsan, and R. D. Sriram. On-toSTEP: OWL-DL ontology for STEP. Technical report, NIST Interagency, 2009.
- [75] C. Lacave and F. J. Diez. A review of explanation methods for heuristic expert systems. *Knowledge Engineering Review*, 19:133–146, 2004.
- [76] Z. Li and B. D’Ambrosio. An efficient approach for finding the mpe in belief networks. In *Proceedings of the Ninth international conference on Uncertainty in artificial intelligence*, pages 342–349, 1993.
- [77] C. Lin, J.-M. Ling, and C.-L. Huang. An expert system for transformer fault diagnosis using dissolved gas analysis. *IEEE Transactions on Power Delivery*, 8:231–238, 1993.
- [78] H. L. Lockett and M. D. Guenov. Graph-based feature recognition for injection moulding based on a mid-surface approach. *Computer-Aided Design*, 37:251–262, 2005.
- [79] T. Lukasiewicz. Expressive probabilistic description logics. *Artificial Intelligence*, 172:852–883, 2008.
- [80] M. Marquez, R. Gill, and A. White. Application of neural networks in feature recognition of mould reinforced plastic parts. *Concurrent Engineering*, 7:115–122, 1999.

- [81] G. R. Mayes. Theories of explanation. In *Internet Encyclopedia of Philosophy*. 2001.
- [82] D. L. McGuinness and F. van Harmelen. OWL web ontology language overview. W3C recommendation, W3C, 2004.
- [83] M. F. McTear. User modelling for adaptive computer systems: a survey of recent developments. *Artificial Intelligence Review*, 7:157–184, 1993.
- [84] J. Mei and Paslaru. Reasoning paradigms for SWRL-enabled ontologies. In *Proceedings of International Workshop on Protege with Rules*, 2005.
- [85] S. E. Middleton, N. R. Shadbolt, and D. C. De Roure. Ontological user profiling in recommender systems. *ACM Transactions on Information and System Security*, 22:54–88, 2004.
- [86] T. Minakawa, Y. Ichikawa, M. Kunugi, K. Shimada, N. Wada, and M. Utsunomiya. Development and implementation of a power system fault diagnosis expert system. *IEEE Transactions on Power Systems*, 10:932–940, 1995.
- [87] V. O. Mittal and C. L. Paris. Generating explanations in context: The system perspective. *Expert Systems with Applications*, 8:491–503, 1995.
- [88] J. D. Moore. A reactive approach to explanation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1504–1510, 1989.



- [89] J. D. Moore and W. R. Swartout. Pointing: A way toward explanation dialogue. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 457–464, 1990.
- [90] B. Motik and U. Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In *Proceedings of the Thirteenth International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, pages 227–241, 2006.
- [91] B. Moulin, H. Irandoust, M. Bélanger, and G. Desbordes. Explanation and argumentation capabilities: Towards the creation of more persuasive agents. *Artificial Intelligence Review*, 17:169–222, 2002.
- [92] R. Neches, W. Swartout, and J. Moore. Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Transactions on Software Engineering*, SE-11:1337 – 1351, 1985.
- [93] D. Nilsson. An efficient algorithm for finding the m most probable configurations in probabilistic expert systems. *Statistics and Computing*, 8:159–173, 1998.
- [94] M. O’connor, H. Knublauch, S. Tu, B. Grosz, M. Dean, W. Grosso, and M. Musen. Supporting rule system interoperability on the semantic web with SWRL. In *Proceedings of the Fourth International Semantic Web Conference*, pages 974–986, 2005.
- [95] A. O’Hagan, C. E. Buck, A. Daneshkhah, J. R. Eiser, P. H. Garthwaite, D. J. Jenkinson, J. E. Oakley, and T. Rakow. *Uncertain Judgements: Eliciting Expert Probabilities*. John Wiley, 2006.

- [96] N. Ozturk and F. Ozturk. Neural network based non-standard feature recognition to integrate CAD and CAM. *Computers in Industry*, 45:123–135, 2001.
- [97] C. L. Paris. Tailoring object descriptions to a user’s level of expertise. *Computational Linguistics*, 14:64–78, 1988.
- [98] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. The Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann Publishers, 1988.
- [99] J. Pearl. *Causality: models, reasoning, and inference*. Cambridge University Press, 2000.
- [100] H. E. Pople. On the mechanization of abductive logic. In *Proceedings of the Third International Joint Conference on Artificial intelligence*, pages 147–152, 1973.
- [101] S. Prabhakar and M. R. Henderson. Automatic form-feature recognition using neural-network-based techniques on boundary representations of solid models. *Computer-Aided Design*, 24:381–393, 1992.
- [102] M. J. Pratt. Introduction to ISO 10303: the STEP standard for product data exchange. *Journal of Computing and Information Science in Engineering*, 1:102–103, 2001.
- [103] V. Rameshbabu and M. S. Shunmugam. Hybrid feature recognition method for setup planning from STEP AP-203. *Robotics and Computer-Integrated Manufacturing*, 25:393–408, 2009.

- [104] E. L. Rissland, E. M. Valcarce, and K. D. Ashley. Explaining and Arguing with Examples. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 288–94, 1984.
- [105] P. S. Sajja. *Knowledge-based systems for socio-economic rural development*. PhD thesis, Sardar Patel University, 2000.
- [106] H. Sakurai. Volume decomposition and feature recognition: part 1 - polyhedral objects. *Computer-Aided Design*, 27:833–843, 1995.
- [107] H. Sakurai. Volume decomposition and feature recognition: part 2 – curved objects. *Computer-Aided Design*, 28:519–537, 1996.
- [108] W. Salmon. *Scientific explanation and the causal structure of the world*. Princeton University Press, 1984.
- [109] W. Salmon. *Four Decades of Scientific Explanation*. University of Pittsburgh Press, 2006.
- [110] W. Salmon, R. Jeffrey, and J. Greeno. *Statistical Explanation & Statistical Relevance*. University of Pittsburgh Press, 1971.
- [111] D. Schenck and P. Wilson. *Information modeling: the EXPRESS way*. Oxford University Press, 1994.
- [112] SCRA. STEP application handbook ISO 10303 version 3. <http://www.uspro.org/>, 2006.

- [113] B. Seroussi and J. Golmard. An algorithm directly finding the k most probable configurations in bayesian networks. *International Journal of Approximate Reasoning*, 11:205 – 233, 1994.
- [114] E. Shortliffe. *Computer-based medical consultations, MYCIN*. Artificial intelligence series. Elsevier, 1976.
- [115] A. Sieg, B. Mobasher, and R. Burke. Ontological user profiles for representing context in web search. In *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*, pages 91–94, 2007.
- [116] E. Sirin and B. Parsia. Optimizations for answering conjunctive abox queries: first results. In *Proceedings of the International Description Logics Workshop*, pages 215–222, 2006.
- [117] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5:51–53, 2007.
- [118] N. Sriwachirawat and S. Auwatanamongkol. On approximating K-MPE of bayesian networks using genetic algorithm. In *Proceedings of the 2006 IEEE Conference on Cybernetics and Intelligent Systems*, pages 1–6, 2006.
- [119] G. Stamou, J. Z. Pan, V. Tzouzavas, and I. Horrocks. A fuzzy extension of SWRL. <http://www.w3.org/2004/12/rules-ws/paper/52/>, 2004.
- [120] I. Stroud. *Boundary representation modelling techniques*. Springer, 2006.

- [121] W. R. Swartout. XPLAIN: a system for creating and explaining expert consulting programs. *Artificial Intelligence*, 21:285–325, 1983.
- [122] P. Thagard. Computational models in the philosophy of science. In *Proceedings of the Biennial Meeting of the Philosophy of Science Association*, pages 329–335, 1986.
- [123] P. Thagard. *Conceptual revolutions*. Princeton paperbacks. Princeton University Press, 1992.
- [124] P. Thagard. *Computational Philosophy of Science*. Bradford Books. MIT Press, 1993.
- [125] P. Thagard. *Coherence in Thought and Action*. Life and Mind: Philosophical Issues in Biology and Psychology. MIT Press, 2002.
- [126] P. Thagard. Causal inference in legal decision making: Explanatory coherence vs. bayesian networks. *Applied Artificial Intelligence*, 18:231–249, 2004.
- [127] P. Thagard. Cognitive science. In *The Stanford Encyclopedia of Philosophy*. 2011.
- [128] P. Thagard and A. Litt. Models of scientific explanation. In *The Cambridge handbook of computational cognitive modeling*. Cambridge University Press, 2006.
- [129] N. Tintarev. Explaining recommendations. In *User Modeling 2007*. Springer Berlin, 2007.

- [130] N. Tintarev. Explanations of recommendations. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 203–206, 2007.
- [131] N. Tintarev and J. Masthoff. A survey of explanations in recommender systems. In *Proceedings of the 23rd International Conference on Data Engineering Workshop*, pages 801–810, 2007.
- [132] X. Tong and J. Ang. Explaining control strategies in second generation expert systems. *IEEE Transactions on Systems, Man and Cybernetics*, 25:1483–1490, 1995.
- [133] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. In *Proceedings of the Third International Joint Conference on Automated Reasoning*, pages 292–297, 2006.
- [134] E. Turban. *Decision support and expert systems (4th ed.): management support systems*. Prentice-Hall, Inc., 1995.
- [135] M. van Lent, W. Fisher, and M. Mancuso. An explainable artificial intelligence system for small-unit tactical behavior. In *Proceedings of the 16th conference on Innovative applications of artificial intelligence*, pages 900–907, 2004.
- [136] W. van Melle. MYCIN: a knowledge-based consultation program for infectious disease diagnosis. *International Journal of Man-Machine Studies*, 10:313 – 322, 1978.

- [137] J. H. Vandenbrande and A. A. G. Requicha. Spatial reasoning for the automatic recognition of machinable features in solid models. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 15:1269–1285, 1993.
- [138] P. K. Venuvinod and S. Y. Wong. A graph-based expert system approach to geometric feature recognition. *Journal of Intelligent Manufacturing*, 6:155–162, 1995.
- [139] J. Vig, S. Sen, and J. Riedl. TAGEXPLANATIONS: explaining recommendations using tags. In *Proceedings of the 14th international conference on Intelligent user interfaces*, pages 47–56, 2009.
- [140] D. Walton. *Abductive reasoning*. University of Alabama Press, 2004.
- [141] Z. Wang, Y. Liu, and P. Griffin. A combined ANN and expert system tool for transformer fault diagnosis. *IEEE Transactions on Power Delivery*, 13:1224–1229, 1998.
- [142] P. Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40:80–91, 1997.
- [143] M. R. Wick. Reconstructive explanation: Explanation as complex problem solving. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 1989.
- [144] M. R. Wick, P. Dutta, T. Wineinger, and J. Conner. Reconstructive explanation: A case study in integral calculus. *Expert Systems with Applications*, 8:463 – 473, 1995.

- [145] M. R. Wick and W. B. Thompson. Reconstructive expert system explanation. *Artificial Intelligence*, 54:33 – 70, 1992.
- [146] T. C. Woo. Feature extraction by volume decomposition. In *Conference on CAD/CAM Technology in Mechanical Engineering*. MIT, 1982.
- [147] J. Woodward. Scientific explanation. In *The Stanford Encyclopedia of Philosophy*. Winter 2011 edition, 2011.
- [148] D. Wu. Active acquisition of user models: Implications for decision-theoretic dialog planning and plan recognition. *User Modeling and User-Adapted Interaction*, 1:149–172, 1991.
- [149] D. Yang, M. Dong, and R. Miao. Development of a product configuration system with an ontology-based approach. *Computer-Aided Design*, 40:863–878, 2008.
- [150] Z. Yongli, H. Limin, and L. Jinling. Bayesian networks-based approach for power systems fault diagnosis. *IEEE Transactions on Power Delivery*, 21:634 – 639, 2006.
- [151] C. Yuan, H. Lim, and T. C. Lu. Most relevant explanation in bayesian networks. *Journal of Artificial Intelligence Research*, 42:309–352, 2011.
- [152] C. Yuan and T. C. Lu. Finding explanations in bayesian networks. In *Proceedings of the 18th International Workshop on Principles of Diagnosis*, 2007.



- [153] X. F. Zha. Planning for STEP-based electro-mechanical assemblies: an integrated approach. *International Journal of Computer Integrated Manufacturing*, 17:305–326, 2004.
- [154] H. L. Zhang, C. V. der Velden, X. Yu, C. Bil, T. Jones, and I. Fieldhouse. Developing a rule engine for automated feature recognition from CAD models. In *Proceedings of the 35th Annual Conference of IEEE on Industrial Electronics*, pages 3925–3930, 2009.
- [155] Y. Zhang, X. Ding, Y. Liu, and P. Griffin. An artificial neural network approach to transformer fault diagnosis. *IEEE Transactions on Power Delivery*, 11:1836–1841, 1996.
- [156] W. Zhao and J. K. Liu. OWL/SWRL representation methodology for EXPRESS-driven product information model. *Computers in Industry*, 59:590–600, 2008.
- [157] G. Zheng and Z. Yongli. Research of transformer fault diagnosis based on bayesian network classifiers. In *Proceedings of the second International Conference on Computer Design and Applications*, pages 382–385, 2010.

# Appendix A

## Recognition rules in OAES

### A.1 Pocket Feature Family

$$\begin{aligned} & \text{face\_set}(?x) \wedge \text{has\_member\_face}(?x, ?a) \wedge \text{has\_member\_face}(?x, ?b) \wedge \text{has\_member\_face}(?x, ?c) \wedge \\ & \quad \text{has\_member\_face}(?x, ?d) \wedge \text{triangular\_pocket\_bottom\_face}(?a) \wedge \text{pocket\_wall\_face}(?b) \wedge \\ & \quad \text{pocket\_wall\_face}(?c) \wedge \text{pocket\_wall\_face}(?d) \wedge \text{face\_has\_neighbor}(?a, ?b) \wedge \\ & \quad \text{face\_has\_neighbor}(?a, ?c) \wedge \text{face\_has\_neighbor}(?a, ?d) \wedge \text{face\_has\_neighbor}(?b, ?c) \wedge \\ & \quad \text{face\_has\_neighbor}(?c, ?d) \wedge \text{face\_has\_neighbor}(?d, ?b) \rightarrow \text{triangular\_pocket}(?x) \end{aligned}$$

**Readable format:** The face\_set is a triangular\_pocket because: the face\_set contains one triangular\_pocket\_bottom\_face (?a) and three pocket\_wall\_faces (?b, ?c, ?d), while the three wall faces are circularly connected with each other and are all connected with the bottom face.

$$\begin{aligned} & \text{face\_set}(?x) \wedge \text{has\_member\_face}(?x, ?a) \wedge \text{has\_member\_face}(?x, ?b) \wedge \text{has\_member\_face}(?x, ?c) \wedge \\ & \quad \text{has\_member\_face}(?x, ?d) \wedge \text{has\_member\_face}(?x, ?e) \wedge \text{quadrangular\_pocket\_bottom\_face}(?a) \wedge \\ & \quad \text{pocket\_wall\_face}(?b) \wedge \text{pocket\_wall\_face}(?c) \wedge \text{pocket\_wall\_face}(?d) \wedge \text{pocket\_wall\_face}(?e) \wedge \end{aligned}$$

$$\begin{aligned}
& face\_has\_neighbor(?a, ?b) \wedge face\_has\_neighbor(?a, ?c) \wedge face\_has\_neighbor(?a, ?d) \wedge \\
& face\_has\_neighbor(?a, ?e) \wedge face\_has\_neighbor(?b, ?c) \wedge face\_has\_neighbor(?c, ?d) \wedge \\
& face\_has\_neighbor(?d, ?e) \wedge face\_has\_neighbor(?d, ?e) \rightarrow quadrangular\_pocket(?x)
\end{aligned}$$

**Readable format:** The `face_set` is a `quadrangular_pocket` because: the `face_set` contains one `quadrangular_pocket_bottom_face` (`?a`) and four `pocket_wall_faces` (`?b`, `?c`, `?d`), while the four wall faces are circularly connected with each other and are all connected with the bottom face.

$$\begin{aligned}
& face\_set(?x) \wedge has\_member\_face(?x, ?a) \wedge has\_member\_face(?x, ?b) \wedge has\_member\_face(?x, ?c) \wedge \\
& has\_member\_face(?x, ?d) \wedge has\_member\_face(?x, ?e) \wedge pocket\_bottom\_face(?a) \wedge \\
& plane\_pocket\_wall\_face(?b) \wedge plane\_pocket\_wall\_face(?d) \wedge curved\_pocket\_wall\_face(?c) \wedge \\
& curved\_pocket\_wall\_face(?e) \wedge face\_has\_neighbor(?a, ?b) \wedge face\_has\_neighbor(?a, ?c) \wedge \\
& face\_has\_neighbor(?a, ?d) \wedge face\_has\_neighbor(?a, ?e) \wedge face\_has\_neighbor(?b, ?c) \wedge \\
& face\_has\_neighbor(?c, ?d) \wedge face\_has\_neighbor(?d, ?e) \wedge face\_has\_neighbor(?d, ?e) \rightarrow \\
& Obround\_pocket(?x)
\end{aligned}$$

**Readable format:** The `face_set` is an `obround_pocket` because: the `face_set` contains one `pocket_bottom_face` (`?a`), two `plane_pocket_wall_faces` (`?b`, `?d`) and two `curved_pocket_wall_faces` (`?c`, `?e`), while the wall faces are connected with each other alternatively and are all connected with the bottom face.

$$face \sqcap \forall y (face\_has\_edge.concave\_angular\_edge \equiv pocket\_bottom\_face);$$

internally translated to:

$$face(?x) \wedge \forall y (face\_has\_edge(?x, ?y) \wedge concave\_angular\_edge(?y)) \rightarrow pocket\_bottom\_face;$$

**Readable format:** ?x is a pocket\_bottom\_face because: ?x is a face whose edges (alternatives of ?y) are all concave\_angular\_edge.

$$triangular\_face(?x) \wedge pocket\_bottom\_face(?x) \rightarrow triangular\_pocket\_bootom\_face(?x)$$

**Readable format:** ?x is a triangular\_pocket\_bottom\_face because: ?x is a triangular\_face as well as a pocket\_bottom\_face.

$$quadrangular\_face(?x) \wedge pocket\_bottom\_face(?x) \rightarrow quadrangular\_pocket\_bootom\_face(?x)$$

**Readable format:** ?x is a quadrangular\_pocket\_bottom\_face because: ?x is a quadrangular\_face as well as a pocket\_bottom\_face.

$$face \sqcap = 1face\_has\_neighbor.pocket\_bottom\_face \equiv pocket\_wall\_face;$$

internally translated to:

$$face(?x) \wedge (?y = 1)(face\_has\_neighbor(?x, ?y) \wedge pocket\_bottom\_face(?y)) \rightarrow pocket\_wall\_face(?x)$$

**Readable format:** ?x is a pocket\_wall\_face because: ?x is a face and only one of the neighbors (?y) of ?x is pocket\_bottom\_face.

$$pocket\_wall\_face(?x) \wedge face\_surface\_has\_face\_geometry(?x, ?y) \wedge plane(?y) \rightarrow \\ plane\_pocket\_wall\_face(?x)$$

**Readable format:** ?x is a plane\_pocket\_wall\_face because: ?x is a pocket\_wall\_face which locates on a plane surface.

$$pocket\_wall\_face(?x) \wedge face\_surface\_has\_face\_geometry(?x, ?y) \wedge cylindrical\_surface(?y) \rightarrow curved\_pocket\_wall\_face(?x)$$

**Readable format:** ?x is a curved\_pocket\_wall\_face because: ?x is a pocket\_wall\_face which locates on a cylindrical surface.

## A.2 Hole Feature Family

$$face\_set(?x) \wedge has\_member\_face(?x, ?a) \wedge has\_member\_face(?x, ?b) \wedge has\_member\_face(?x, ?c) \wedge hole\_wall\_face(?a) \wedge hole\_wall\_face(?b) \wedge hole\_wall\_face(?c) \wedge face\_has\_neighbor(?a, ?b) \wedge face\_has\_neighbor(?b, ?c) \wedge face\_has\_neighbor(?c, ?a) \rightarrow triangular\_hole(?x)$$

**Readable format:** The face\_set is a triangular\_hole because: the face\_set contains three hole\_wall\_faces (?a, ?b, ?c), while the three wall faces are circularly connected with each other.

$$face\_set(?x) \wedge has\_member\_face(?x, ?a) \wedge has\_member\_face(?x, ?b) \wedge has\_member\_face(?x, ?c) \wedge face\_has\_face(?x, ?d) \wedge hole\_wall\_face(?a) \wedge hole\_wall\_face(?b) \wedge hole\_wall\_face(?c) \wedge hole\_wall\_face(?d) \wedge face\_has\_neighbor(?a, ?b) \wedge face\_has\_neighbor(?b, ?c) \wedge face\_has\_neighbor(?c, ?d) \wedge face\_has\_neighbor(?d, ?a) \rightarrow quadrangular\_hole(?x)$$

**Readable format:** The face\_set is a quadrangular\_hole because: the face\_set contains four hole\_wall\_faces (?a, ?b, ?c, ?d), while the four wall faces are circularly connected with each other.

$$\begin{aligned}
& face\_set(?x) \wedge has\_member\_face(?x, ?a) \wedge has\_member\_face(?x, ?b) \wedge has\_member\_face(?x, ?c) \wedge \\
& \quad face\_has\_face(?x, ?d) \wedge plane\_hole\_wall\_face(?a) \wedge curved\_hole\_wall\_face(?b) \wedge \\
& \quad plane\_hole\_wall\_face(?c) \wedge curved\_hole\_wall\_face(?d) \wedge face\_has\_neighbor(?a, ?b) \wedge \\
& \quad face\_has\_neighbor(?b, ?c) \wedge face\_has\_neighbor(?c, ?d) \wedge face\_has\_neighbor(?d, ?a) \rightarrow \\
& \quad obround\_hole(?x)
\end{aligned}$$

**Readable format:** The face\_set is an obround\_hole because: the face\_set contains two plane\_hole\_wall\_faces (?a, ?c) and two curved\_hole\_wall\_face (?b, ?d), and the two types of wall faces are connected with each other alternatively.

$$\begin{aligned}
& face(?x) \wedge face\_inner\_bound(?m) \wedge face\_inner\_bound(?n) \wedge face\_has\_edge(?x, ?y) \wedge \\
& face\_has\_edge(?x, ?z) \wedge face\_bound\_has\_edge(?m, ?y) \wedge face\_bound\_has\_edge(?n, ?z) \wedge \\
& \quad differentFrom(?m, ?n) \wedge differentFrom(?y, ?z) \rightarrow hole\_wall\_face(?x)
\end{aligned}$$

**Readable format:** ?x is a hole\_wall\_face because: it is a face which has two different edges ?y and ?z, and the two edges belong to two different face\_inner\_bound ?m and ?n respectively.

$$\begin{aligned}
& hole\_wall\_face(?x) \wedge face\_surface\_has\_face\_geometry(?x, ?y) \wedge plane(?y) \rightarrow \\
& \quad plane\_hole\_wall\_face(?x)
\end{aligned}$$

**Readable format:** ?x is a plane\_hole\_wall\_face because: ?x is a hole\_wall\_face which locates on a plane surface.

$$\begin{aligned}
& hole\_wall\_face(?x) \wedge face\_surface\_has\_face\_geometry(?x, ?y) \wedge cylindrical\_surface(?y) \rightarrow \\
& \quad curved\_hole\_wall\_face(?x)
\end{aligned}$$

**Readable format:** ?x is a curved\_hole\_wall\_face because: ?x is a hole\_wall\_face which locates on a cylindrical surface.

### A.3 Blind Step Feature Family

$$face\_set(?x) \wedge has\_member\_face(?x, ?a) \wedge has\_member\_face(?x, ?b) \wedge blind\_step\_bottom\_face(?a) \wedge plane\_blind\_step\_wall\_face(?b) \wedge face\_has\_concave\_neighbor(?a, ?b) \rightarrow chamfered\_blind\_step(?x)$$

**Readable format:** The face\_set is a chamfered\_blind\_step because: the face\_set contains one blind\_step\_bottom\_face (?a) and one plane\_blind\_step\_wall\_face (?b), and the two faces are concavely connected with each other.

$$face\_set(?x) \wedge has\_member\_face(?x, ?a) \wedge has\_member\_face(?x, ?b) \wedge blind\_step\_bottom\_face(?a) \wedge curved\_blind\_step\_wall\_face(?b) \wedge face\_has\_concave\_neighbor(?a, ?b) \rightarrow circular\_blind\_step(?x)$$

**Readable format:** The face\_set is a circular\_blind\_step because: the face\_set contains one blind\_step\_bottom\_face (?a) and one curved\_blind\_step\_wall\_face (?b), and the two faces are concavely connected with each other.

$$face\_set(?x) \wedge has\_member\_face(?x, ?a) \wedge has\_member\_face(?x, ?b) \wedge has\_member\_face(?x, ?c) \wedge blind\_step\_bottom\_face(?a) \wedge plane\_blind\_step\_wall\_face(?b) \wedge plane\_blind\_step\_wall\_face(?c) \wedge face\_has\_concave\_neighbor(?a, ?b) \wedge face\_has\_concave\_neighbor(?a, ?c) \wedge face\_concave\_has\_neighbor(?b, ?c) \rightarrow quadrangular\_blind\_step(?x)$$

**Readable format:** The face\_set is a quadrangular\_blind\_step because: the face\_set contains one blind\_step\_bottom\_face (?a) and two plane\_blind\_step\_wall\_face (?b, ?c), and the three faces are all concavely connected with each other.

$$\begin{aligned}
& \text{face}(?x) \wedge (\text{face\_has\_convex\_neighbor} = 2)(?x, ?y) \wedge \text{face\_has\_convex\_neighbor}(?x, ?m) \wedge \\
& \text{face\_has\_convex\_neighbor}(?x, ?n) \wedge \text{face\_has\_convex\_neighbor}(?m, ?n) \rightarrow \\
& \text{blind\_step\_bottom\_face}(?x)
\end{aligned}$$

**Readable format:** ?x is a blind\_step\_bottom\_face because: ?x is a face which has exactly two convex neighbors, and these two neighbor faces connect with each other in a convex angle.

$$\text{face} \sqcap = 1 \text{face\_has\_concave\_neighbor.blind\_step\_bottom\_face} \equiv \text{blind\_slot\_wall\_face};$$

internally translated to:

$$\begin{aligned}
& \text{face}(?x) \wedge (?y = 1)(\text{face\_has\_concave\_neighbor}(?x, ?y) \wedge \text{blind\_step\_bottom\_face}(?y)) \rightarrow \\
& \text{blind\_slot\_wall\_face}(?x)
\end{aligned}$$

**Readable format:** ?x is a blind\_step\_wall\_face because: ?x is a face which is concavely connected with one blind\_step\_bottom\_face ?y only.

$$\begin{aligned}
& \text{blind\_slot\_wall\_face}(?x) \wedge \text{face\_surface\_has\_face\_geometry}(?x, ?y) \wedge \text{plane}(?y) \rightarrow \\
& \text{plane\_blind\_slot\_wall\_face}(?x)
\end{aligned}$$

**Readable format:** ?x is a plane\_blind\_slot\_wall\_face because: ?x is a blind\_slot\_wall\_face which locates on a plane surface.

$$\begin{aligned}
& \text{blind\_slot\_wall\_face}(?x) \wedge \text{face\_surface\_has\_face\_geometry}(?x, ?y) \wedge \text{cylindrical\_surface}(?y) \rightarrow \\
& \text{curved\_blind\_slot\_wall\_face}(?x)
\end{aligned}$$

**Readable format:** ?x is a curved\_blind\_slot\_wall\_face because: ?x is a blind\_slot\_wall\_face(?x) which locates on a cylindrical surface.



## A.4 Through Step Feature Family

$$\begin{aligned}
 & \text{face\_set}(?x) \wedge \text{has\_member\_face}(?x, ?a) \wedge \text{has\_member\_face}(?x, ?b) \wedge \\
 & \text{through\_step\_bottom\_face}(?a) \wedge \text{through\_step\_wall\_face}(?b) \wedge \text{face\_has\_concave\_neighbor}(?a, ?b) \rightarrow \\
 & \text{regular\_through\_step}(?x)
 \end{aligned}$$

**Readable format:** The `face_set` is a `regular_through_step` because: the `face_set` contains one `through_step_bottom_face` (?a) and one `through_step_wall_face` (?b), and the two faces are connected with each other.

$$\begin{aligned}
 & \text{face\_set}(?x) \wedge \text{has\_member\_face}(?x, ?a) \wedge \text{has\_member\_face}(?x, ?b) \wedge \text{has\_member\_face}(?x, ?c) \wedge \\
 & \text{through\_step\_bottom\_face}(?a) \wedge \text{through\_step\_wall\_face}(?b) \wedge \text{through\_step\_wall\_face}(?c) \wedge \\
 & \text{face\_has\_neighbor}(?a, ?b) \wedge \text{face\_has\_neighbor}(?a, ?c) \wedge \text{face\_has\_concave\_neighbor}(?b, ?c) \rightarrow \\
 & \text{2side\_concave\_through\_step}
 \end{aligned}$$

**Readable format:** The `face_set` is a `2side_concave_through_step` because: the `face_set` contains one `through_step_bottom_face` (?a) and two `through_step_wall_faces` (?b, ?c), while all the faces are connected with each other and the two wall faces are concavely connected.

$$\begin{aligned}
 & \text{face\_set}(?x) \wedge \text{has\_member\_face}(?x, ?a) \wedge \text{has\_member\_face}(?x, ?b) \wedge \text{has\_member\_face}(?x, ?c) \wedge \\
 & \text{through\_step\_bottom\_face}(?a) \wedge \text{through\_step\_wall\_face}(?b) \wedge \text{through\_step\_wall\_face}(?c) \wedge \\
 & \text{face\_has\_neighbor}(?a, ?b) \wedge \text{face\_has\_neighbor}(?a, ?c) \wedge \text{face\_has\_convex\_neighbor}(?b, ?c) \rightarrow \\
 & \text{2side\_convex\_through\_step}
 \end{aligned}$$

**Readable format:** The `face_set` is a `2side_convex_through_step` because: the `face_set` contains one `through_step_bottom_face` (?a) and two `through_step_wall_faces` (?b, ?c),

while all the faces are connected with each other and the two wall faces are convexly connected.

$$\begin{aligned}
& \text{face}(?x) \wedge (\text{face\_has\_convex\_neighbor} = 3)(?x, ?y) \wedge \text{face\_has\_convex\_neighbor}(?x, ?a) \wedge \\
& \quad \text{face\_has\_convex\_neighbor}(?x, ?b) \wedge \text{face\_has\_convex\_neighbor}(?x, ?c) \wedge \\
& \quad \text{face\_has\_convex\_neighbor}(?a, ?b) \wedge \text{face\_has\_convex\_neighbor}(?b, ?c) \wedge \text{swrlb} : \\
& \quad \text{opposite\_direction\_check}(?a, ?c) \rightarrow \text{through\_step\_bottom\_face}(?x)
\end{aligned}$$

**Readable format:** ?x is a through\_step\_bottom\_face because: ?x is a face which has exactly three convex neighbors ?a, ?b and ?c, and the three neighbor faces convexly connect with each other in series while two side faces ?a and ?c have opposite surface direction.

$$\text{face} \sqcap = 1 \text{face\_has\_concave\_neighbor.through\_step\_bottom\_face} \equiv \text{through\_step\_wall\_face};$$

internally translated to:

$$\begin{aligned}
& \text{face}(?x) \wedge (?y = 1)(\text{face\_has\_concave\_neighbor}(?x, ?y) \wedge \text{through\_step\_bottom\_face}(?y)) \rightarrow \\
& \quad \text{through\_step\_wall\_face}(?x)
\end{aligned}$$

**Readable format:** ?x is a through\_step\_wall\_face because: ?x is a face which is concavely connected with one through\_step\_bottom\_face ?y only.

## A.5 Blind Slot Feature Family

$$\begin{aligned}
& \text{face\_set}(?x) \wedge \text{has\_member\_face}(?x, ?a) \wedge \text{has\_member\_face}(?x, ?b) \wedge \text{has\_member\_face}(?x, ?c) \wedge \\
& \quad \text{has\_member\_face}(?x, ?d) \wedge \text{blind\_slot\_bottom\_face}(?a) \wedge \text{blind\_slot\_wall\_face}(?b) \wedge
\end{aligned}$$

$$\begin{aligned}
& blind\_slot\_wall\_face(?c) \wedge blind\_slot\_wall\_face(?d) \wedge face\_has\_neighbor(?a, ?b) \wedge \\
& face\_has\_neighbor(?a, ?c) \wedge face\_has\_neighbor(?a, ?d) \wedge face\_has\_neighbor(?b, ?c) \wedge \\
& face\_has\_neighbor(?c, ?d) \rightarrow regular\_blind\_slot(?x)
\end{aligned}$$

**Readable format:** The face\_set is a regular\_blind\_slot because: the face\_set contains one blind\_slot\_bottom\_face(?a) and three blind\_slot\_wall\_face (?b, ?c, ?d), while the three wall faces are connected with each other and are all connected with the bottom face.

$$\begin{aligned}
& face\_set(?x) \wedge has\_member\_face(?x, ?a) \wedge has\_member\_face(?x, ?b) \wedge has\_member\_face(?x, ?c) \wedge \\
& has\_member\_face(?x, ?d) \wedge blind\_slot\_bottom\_face(?a) \wedge plane\_blind\_slot\_wall\_face(?b) \wedge \\
& curved\_blind\_slot\_wall\_face(?c) \wedge plane\_blind\_slot\_wall\_face(?d) \wedge face\_has\_neighbor(?a, ?b) \wedge \\
& face\_has\_neighbor(?a, ?c) \wedge face\_has\_neighbor(?a, ?d) \wedge face\_has\_neighbor(?b, ?c) \wedge \\
& face\_has\_neighbor(?c, ?d) \rightarrow vertical\_obround\_blind\_slot(?x)
\end{aligned}$$

**Readable format:** The face\_set is a regular\_blind\_slot because: the face\_set contains one blind\_slot\_bottom\_face(?a), one curved\_blind\_slot\_wall\_face (?c), and two plane\_blind\_slot\_wall\_face (?b, ?d), while the three wall faces are connected together with the curved wall face in the middle, and are all connected with the bottom face.

$$\begin{aligned}
& face\_set(?x) \wedge has\_member\_face(?x, ?a) \wedge has\_member\_face(?x, ?b) \wedge has\_member\_face(?x, ?c) \wedge \\
& has\_member\_face(?x, ?d) \wedge blind\_slot\_bottom\_face(?a) \wedge curved\_blind\_slot\_wall\_face(?b) \wedge \\
& plane\_blind\_slot\_wall\_face(?c) \wedge curved\_blind\_slot\_wall\_face(?d) \wedge face\_has\_neighbor(?a, ?b) \wedge \\
& face\_has\_neighbor(?a, ?c) \wedge face\_has\_neighbor(?a, ?d) \wedge face\_has\_neighbor(?b, ?c) \wedge \\
& face\_has\_neighbor(?c, ?d) \rightarrow vertical\_obround\_blind\_slot(?x)
\end{aligned}$$

**Readable format:** The `face_set` is a `regular_blind_slot` because: the `face_set` contains one `blind_slot_bottom_face(?a)`, one `plane_blind_slot_wall_face (?c)`, and two `curved_blind_slot_wall_face (?b, ?d)`, while the three wall faces are connected together with the plane wall face in the middle, and are all connected with the bottom face.

$$face(?x) \wedge (face\_has\_convex\_neighbor = 1)(?x, ?y) \rightarrow blind\_slot\_bottom\_face(?x);$$

**Readable format:** `?x` is a `blind_slot_bottom_face` because: `?x` is a face which has only one convex neighbor.

$$face \sqcap = 1 face\_has\_neighbor.blind\_slot\_bottom\_face \equiv blind\_slot\_wall\_face;$$

internally translated to:

$$face(?x) \wedge (?y = 1)(face\_has\_neighbor(?x, ?y) \wedge blind\_slot\_bottom\_face(?y)) \rightarrow \\ blind\_slot\_wall\_face(?x)$$

**Readable format:** `?x` is a `blind_slot_wall_face` because: `?x` is a face which is a neighbor of one `blind_slot_bottom_face ?y` only.

## A.6 Through Slot Feature Family

$$face\_set(?x) \wedge has\_member\_face(?x, ?a) \wedge circular\_through\_slot\_face(?a) \rightarrow \\ circular\_through\_slot(?x)$$

**Readable format:** The `face_set` is a `circular_through_slot` because: the `face_set` contains only one `circular_through_slot_face(?a)`.

$$\begin{aligned}
& face\_set(?x) \wedge has\_member\_face(?x, ?a) \wedge has\_member\_face(?x, ?b) \wedge V\_through\_slot\_face(?a) \wedge \\
& V\_through\_slot\_face(?b) \wedge face\_has\_concave\_neighbor(?a, ?b) \wedge face\_has\_convex\_neighbor(?a, ?m) \wedge \\
& face\_has\_convex\_neighbor(?b, ?n) \wedge differentFrom(?m, ?n) \wedge swrlb : \\
& same\_direction\_check(?m, ?n) \rightarrow V\_through\_slot(?x)
\end{aligned}$$

**Readable format:** The face\_set is a V\_through\_slot because: the face\_set contains two V\_through\_slot\_face(?a, ?b) which are concavely connected with each other, and they each have a concave neighbor which have same surface direction.

$$\begin{aligned}
& face\_set(?x) \wedge has\_member\_face(?x, ?a) \wedge has\_member\_face(?x, ?b) \wedge has\_member\_face(?x, ?c) \wedge \\
& through\_slot\_bottom\_face(?a) \wedge through\_slot\_wall\_face(?b) \wedge through\_slot\_wall\_face(?c) \wedge \\
& face\_has\_concave\_neighbor(?a, ?b) \wedge face\_has\_concave\_neighbor(?b, ?c) \rightarrow \\
& regular\_through\_slot(?x)
\end{aligned}$$

**Readable format:** The face\_set is a regular\_through\_slot because: the face\_set contains one through\_slot\_bottom\_face(?a) and the two through\_slot\_wall\_faces (?b, ?c), while the three faces are concave connected in series with the bottom face in the middle.

$$\begin{aligned}
& quadrangular\_face(?x) \wedge face\_surface\_has\_face\_geometry(?x, ?y) \wedge cylindrical\_surface(?y) \wedge \\
& face\_has\_convex\_neighbor(?x, ?a) \wedge face\_has\_convex\_neighbor(?x, ?b) \wedge \\
& face\_has\_convex\_neighbor(?x, ?c) \wedge face\_has\_convex\_neighbor(?x, ?d) \wedge swrlb : \\
& opposite\_direction\_check(?a, ?c) \wedge swrlb : same\_direction\_check(?b, ?d) \rightarrow \\
& circular\_through\_slot\_face(?x)
\end{aligned}$$

**Readable format:** ?x looks like a circular\_through\_slot\_face because: ?x is a quadrangular face which locates on a cylindrical\_surface ?y, and ?x has four convex neigh-

bors ?a, ?b, ?c and ?d, in which two faces (?a, ?c) have opposite surface directions while the other two (?b, ?d) have same surface direction.

$$\begin{aligned} & quadrangular\_face(?x) \wedge (face\_has\_concave\_neighbor = \\ & 1)(?x, ?a) \wedge face\_has\_convex\_neighbor(?x, ?b) \wedge face\_has\_convex\_neighbor(?x, ?c) \wedge swrlb : \\ & opposite\_direction\_check(?b, ?c) \rightarrow V\_through\_slot\_face(?x) \end{aligned}$$

**Readable format:** ?x looks like a circular\_through\_slot\_face because: ?x is a quadrangular face which has only one concave neighbor ?a, and two of ?x's convex neighbor ?b, ?c have opposite surface directions.

$$\begin{aligned} & quadrangular\_face(?x) \wedge face\_has\_convex\_neighbor(?x, ?a) \wedge face\_has\_concave\_neighbor(?x, ?b) \wedge \\ & face\_has\_convex\_neighbor(?x, ?c) \wedge face\_has\_conconcave\_neighbor(?x, ?d) \wedge swrlb : \\ & opposite\_direction\_check(?a, ?c) \wedge swrlb : opposite\_direction\_check(?b, ?d) \rightarrow \\ & through\_slot\_bottom\_face(?x); \end{aligned}$$

**Readable format:** ?x is a through\_slot\_bottom\_face because: ?x is a quadrangular face which has a pair of convex neighbor (?a, ?c) and a pair of concave neighbor (?b, ?d), and in both pair the partners have opposite surface directions.

$$face \sqcap = 1 face\_has\_neighbor.through\_slot\_bottom\_face \equiv through\_slot\_wall\_face;$$

internally translated to:

$$\begin{aligned} & face(?x) \wedge (?y = 1)(face\_has\_neighbor(?x, ?y) \wedge through\_slot\_bottom\_face(?y)) \rightarrow \\ & through\_slot\_wall\_face(?x) \end{aligned}$$

**Readable format:** ?x is a through\_slot\_wall\_face because: ?x is a face which is a neighbor of one through\_slot\_bottom\_face ?y only.

## A.7 Basic constructs

$$face \sqcap \exists face\_surface\_has\_face\_geometry.plane \sqcap = 3face\_has\_edge \equiv triangular\_face;$$

internally translated to:

$$face(?x) \wedge face\_surface\_has\_face\_geometry(?x, ?y) \wedge plane(?y) \wedge (face\_has\_edge = 3)(?x, ?e) \rightarrow \\ triangular\_face(?x)$$

**Readable format:** ?x is a triangular\_face because: ?x is a face which locates on a plane surface ?y and has exactly three edges (alternatives of ?e).

$$face \sqcap \exists face\_surface\_has\_face\_geometry.plane \sqcap = 4face\_has\_edge \equiv triangular\_face;$$

internally translated to:

$$face(?x) \wedge face\_surface\_has\_face\_geometry(?x, ?y) \wedge plane(?y) \wedge (face\_has\_edge = 4)(?x, ?e) \rightarrow \\ triangular\_face(?x)$$

**Readable format:** ?x is a quadrangular\_face because: ?x is a face which locates on a plane surface ?y and has exactly four edges (alternatives of ?e).

$$face(?x) \wedge face(?y) \wedge face\_has\_edge(?x, ?e) \wedge face\_has\_edge(?y, ?e) \wedge differentFrom(?x, ?y) \wedge swrlb : \\ concaveCheck(?x, ?y, ?e) \rightarrow concave\_angular\_edge(?e);$$

**Readable format:** ?e is a concave\_angular\_edge because: ?e is the intersection edge of two different faces ?x and ?y which intersect in a concave angle.

$$face(?x) \wedge face(?y) \wedge face\_has\_edge(?x, ?e) \wedge face\_has\_edge(?y, ?e) \wedge differentFrom(?x, ?y) \rightarrow \\ face\_has\_neighbor(?x, ?y)$$

**Readable format:** face ?x and face ?y are neighbors because: they have a same edge ?e.

$$face(?x) \wedge face(?y) \wedge face\_has\_edge(?x, ?e) \wedge face\_has\_edge(?y, ?e) \wedge differentFrom(?x, ?y) \wedge swrlb : \\ concaveCheck(?x, ?y, ?e) \rightarrow face\_has\_concave\_neighbor(?x, ?y);$$

**Readable format:** face ?x and face ?y are concave neighbors because: they have a same edge ?e and they intersect in a concave angle.

$$face(?x) \wedge face(?y) \wedge face\_has\_edge(?x, ?e) \wedge face\_has\_edge(?y, ?e) \wedge differentFrom(?x, ?y) \wedge swrlb : \\ convexCheck(?x, ?y, ?e) \rightarrow face\_has\_convex\_neighbor(?x, ?y);$$

**Readable format:** face ?x and face ?y are convex neighbors because: they have a same edge ?e and they intersect in a convex angle.

$$face(?x) \wedge face\_has\_face\_bounds(?x, ?y) \wedge face\_bound\_has\_bound(?y, ?z) \wedge \\ edge\_loop\_has\_edges(?z, ?n) \wedge oriented\_edge\_has\_edge\_element(?m, ?n) \rightarrow face\_has\_edge(?x, ?n);$$

**Readable format:** ?n is an edge of face ?x because: ?x has a face\_bound ?y, and ?y locates on an edge\_loop ?z, and ?z contains an oriented\_edge ?m, and finally ?m locates on the edge ?n.



$$\begin{aligned}
& face(?x) \wedge face\_has\_face\_bounds(?x, ?y) \wedge face\_outer\_bound(?y) \wedge face\_bound\_has\_bound(?y, ?z) \wedge \\
& \quad edge\_loop\_has\_edges(?z, ?n) \wedge oriented\_edge\_has\_edge\_element(?m, ?n) \rightarrow \\
& \quad face\_has\_outer\_edge(?x, ?n);
\end{aligned}$$

**Readable format:** ?n is an outer edge of face ?x because: ?x has a face\_outer\_bound ?y, and ?y locates on an edge\_loop ?z, and ?z contains an oriented\_edge ?m, and finally ?m locates on the edge ?n.