

# Sheffield Hallam University

*A multi-agent approach to adaptive learning using a structured ontology classification system*

EHIMWENMA, Kennedy Efosa

Available from the Sheffield Hallam University Research Archive (SHURA) at:

<http://shura.shu.ac.uk/18747/>

## A Sheffield Hallam University thesis

This thesis is protected by copyright which belongs to the author.

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author.

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.

Please visit <http://shura.shu.ac.uk/18747/> and <http://shura.shu.ac.uk/information.html> for further details about copyright and re-use permissions.

**A MULTI-AGENT APPROACH TO ADAPTIVE  
LEARNING USING A STRUCTURED ONTOLOGY  
CLASSIFICATION SYSTEM**

**EHIMWENMA, K. E.      Ph.D.      2017**

A Multi-Agent Approach to Adaptive Learning using a  
Structured Ontology Classification System

Kennedy Efosa Ehimwenma

A Thesis submitted in Fulfilment for the Requirements  
of Degree of Doctor of Philosophy  
Sheffield Hallam University  
United Kingdom

December 2017

## **Abstract**

Diagnostic assessment is an important part of human learning. Tutors in face-to-face classroom environment evaluate students' prior knowledge before the start of a relatively new learning. In that perspective, this thesis investigates the development of an-agent based Pre-assessment System in the identification of knowledge gaps in students' learning between a student's desired concept and some prerequisites concepts. The aim is to test a student's prior skill before the start of the student's higher and desired concept of learning. This thesis thus presents the use of Prometheus agent based software engineering methodology for the Pre-assessment System requirement specification and design. Knowledge representation using a description logic TBox and ABox for defining a domain of learning. As well as the formal modelling of classification rules using rule-based approach as a reasoning process for accurate categorisation of students' skills and appropriate recommendation of learning materials. On implementation, an agent oriented programming language whose facts and rule structure are prolog-like was employed in the development of agents' actions and behaviour. Evaluation results showed that students have skill gaps in their learning while they desire to study a higher-level concept at a given time.

## **Dedication**

To my lovely wife and children

## Acknowledgment

The journey of a PhD is from *Point A to B*. Whereas you know point A, you can never guess the *Point B* nor how to get there. This is because the route to *Point B* is multifaceted: There is no absolute route. To have been able to reach the *Point B*, my deep and profound gratitude goes to God Almighty for the strength and wisdom. This wouldn't have been possible without my supervisory team—Dr Paul Crowther (Director of Studies) and Dr Martin Beer (Supervisor). I thank you for your immense support, guidance, collaboration, contributions and professionalism in seeing to the successful and logical completion of this research study. Dear sirs, you are not only my mentors, but my inspiration and models to emulate. Also worthy of my appreciation is Professor A. U. Osunde (University of Benin) and Professor Maggie McPherson (University of Leeds) who gave me faith and recommended me for this study. I will not fail to mention Professor Jomi Hubner, Department of Automation and Systems Engineering, Federal University of Santa Catarina, Brazil for his technical advice on Jason programming syntax. My sincere appreciation also goes to Mr Caleb Adedigba for his encouragement and psychological boost throughout the period of this work. To my lovely wife Esohe, I owe huge indebtedness for her show of understanding, finance and moral support for the arrival of this moment. Also not left out are my beloved children—Destiny Osawese, Wellington Iyosayi and Praise Isiuwa—who lost most of their playtime with ‘daddy’ to the study-time of this research—that “*daddy is always studying, when are you going to take a break?*” Kids I say to you; *you’ve got me back now*. Lastly, my recognition goes to my mother Mrs Mary Ehimwenma for her relentless prayers and others too numerous to mention that encouraged the successful completion of this research degree.

# Table of Contents

Abstract .....	iii
Dedication .....	iv
Acknowledgment.....	v
Table of Contents .....	vi
List of Figures .....	xiii
List of Tables.....	xviii
Abbreviations .....	xix
Glossary.....	xxi
<b>Chapter 1 .....</b>	<b>1</b>
<b>Introduction and Pre-Learning Diagnosis .....</b>	<b>1</b>
1. Introduction .....	1
1.1 Motivation for Study .....	1
1.2 Research Question.....	2
1.3 Purpose of The Research .....	2
1.4 Aim of The Study .....	2
1.5 Objectives of The Study .....	2
1.6 Defining The Pre-assessment System .....	3
1.7 What is Learning? .....	3
1.7.1 Classification of Students' Learning.....	4
1.7.2 Human Learning .....	4
1.8 Need for Pre-assessment in Learning.....	6
1.9 Contribution to Knowledge .....	8
1.10 Overview of Thesis .....	9
1.11 Publications from this Work.....	9
<b>Chapter 2 .....</b>	<b>11</b>
<b>Knowledge Representation and Intelligent Tutoring Systems.....</b>	<b>11</b>
2. Introduction .....	11
2.1 Knowledge Representation and Ontology.....	11
2.2 Description Logic and Ontology Languages.....	12
2.2.1 SHOE: Simple HTML Ontology Extension .....	12
2.2.2 DAML-ONT: DARPA Agent Markup Language-ONTology.....	12

2.2.3 OIL: Ontology Inference Layer .....	13
2.2.4 DAML+OIL.....	13
2.2.5 RDF: Resource Description Framework.....	14
2.2.6 RDFS : Resource Description Framework Schema.....	15
2.2.7 OWL .....	15
2.3 TBox Terminology .....	16
2.4 ABox World Description.....	19
2.5 Answer Sets Prolog .....	19
2.6 Classification .....	22
2.7 Condition-Action Rule .....	22
2.8 Intelligent Tutoring and Learning Systems .....	23
2.9 SQL Assessment and Learning System.....	24
2.10 Chunking: An Educational Theory of Learning.....	25
2.11 Approaches to Agent Based Learning and Formative Assessment Systems .	26
2.12 Recommender Systems in Education .....	29
2.13 Student Modelling .....	30
2.13.1 Traditional Three-Model.....	31
2.13.2 Classical Four-Model.....	31
2.13.3 New-Generation Architectures .....	32
2.14 Summary of Chapter.....	32
<b>Chapter 3 .....</b>	<b>35</b>
<b>Agents, Agent Oriented Methodologies and Interaction .....</b>	<b>35</b>
3. Introduction .....	35
3.1 Agents.....	35
3.2 Properties of Agent.....	37
3.3 Agent Architectures.....	37
3.3.1 Logic-based Architecture.....	38
3.3.2 Reactive Architecture .....	38
3.3.3 Hybrid Architecture .....	38
3.3.4 BDI Architecture.....	39
3.4 Agent Oriented Methodologies .....	40
3.4.1 Gaia.....	40



3.4.2 Tropos .....	42
3.4.3 Prometheus.....	43
3.5 Comparison of AOSE Methodology .....	46
3.6 The Speech Acts Theory .....	47
3.6.1 John Austin: 1962 .....	47
3.6.2 John Searle: 1969.....	49
3.7 Pre, Post & Completion Conditions .....	50
3.8 Agent Communication Languages .....	50
3.9 Agent Oriented Programming languages and Platforms .....	51
3.9.1 Agent0.....	52
3.9.2 PLACA .....	52
3.9.3 GOAL .....	52
3.9.4 Soar .....	52
3.9.5 JACK .....	53
3.9.6 Jadex .....	53
3.9.7 Jade .....	54
3.9.8 AgentSpeak.....	54
3.9.9 Jason Agent Language .....	54
3.10 Agent Interaction in Jason .....	56
3.10.1 Beliefs .....	57
3.10.2 Annotations.....	57
3.10.3 Goals .....	58
3.10.4 Mental Notes.....	59
3.10.5 Internal Actions.....	59
3.10.6 Plan .....	60
3.10.7 Why Jason Agent Language?.....	61
3.11 Agent Environment Programming .....	62
3.11.1 Artifacts and Human Interaction.....	62
3.11.2 The CArtAgO Artifact .....	63
3.12 Summary of Chapter.....	63
<b>Chapter 4 .....</b>	<b>65</b>
<b>Methodology: Agent Oriented Analysis &amp; Design and Classification Method ..</b>	<b>65</b>

4.	Introduction .....	65
4.1	Prometheus Agent Oriented Software Engineering .....	65
4.1.1	Notation Symbols of PDT.....	66
4.2	System Specification .....	67
4.2.1	Scenario Overview.....	70
4.2.2	System Goal Diagram.....	70
4.2.3	Set of Functionalities .....	71
4.3	Architectural Design.....	72
4.3.1	Analysis Overview.....	72
4.3.2	Agent Role Ordering.....	73
4.3.3	System Overview .....	74
4.4	Detailed Design .....	77
4.4.1	Agent Overview .....	77
4.4.2	Roles and Capability Descriptors for Agents .....	81
4.5	The Student Model .....	83
4.6	The Pre-assessment Mechanism.....	86
4.7	The Learner Component.....	87
4.8	Pre-assessment By Immediate Next Prerequisite Class .....	89
4.8.1	Logic Based Classification Specification for Pre-assessment in a Regular Ontology Model.....	89
4.9	Pre-assessment By Multiple Prerequisite Classes .....	93
4.9.1	Logic Based Classification Specification for Pre-assessment in a Non- Regular Ontology Model .....	93
4.9.2	Estimating The Number of Rules by Prerequisites $Ci, j$ and Leafnodes $Nj, k$ Notation in a Tree .....	99
4.10	Summary of Chapter.....	102
	<b>Chapter 5 .....</b>	<b>103</b>
	<b>A SQL Ontology and The Pre-assessment System.....</b>	<b>103</b>
5.	Introduction .....	103
5.1	Contextual Learning Structure .....	103
5.2	Description Logic for SQL Ontology.....	104
5.2.1	TBox Description for a SQL Ontology.....	105

5.2.2 SQL Individuals in Description Language .....	107
5.2.3 ABox Assertion for a SQL Ontology .....	107
5.3 Digraph analysis of the Description Logic SQL Ontology Model.....	108
5.3.1 A Regular SQL Ontology .....	109
5.3.2 Non-Regular SQL Ontology Model .....	109
5.4 Navigation of Ontology Nodes.....	112
5.5 Ontology Building Tools: Jena API and Protégé Ontology Editor .....	115
5.5.1 Constructing ontologies in Jena API .....	115
5.5.2 Protégé Ontology Tool.....	116
5.6 The Pre-assessment System.....	119
5.6.1 CArtAgO + Jason .....	120
5.7 The Pre-assessment System Environment.....	121
5.8 Programming CArtAgO for Open-Ended Percepts.....	122
5.9 The Agents of the Pre-assessment System .....	123
5.9.1 Agent <i>agInterface</i> and Percept Observation.....	123
5.9.2 Agent <i>agModelling</i> and Classification .....	125
5.9.3 Agent <i>agModel</i> and Student History .....	130
5.9.4 The Agent <i>agSupport</i> and Pre-assessment.....	131
5.9.5 Agent <i>agMaterial</i> and Ontology.....	133
5.10 Summary of Chapter.....	135
<b>Chapter 6 .....</b>	<b>137</b>
<b>System Evaluation, Results and Analysis of Data .....</b>	<b>137</b>
6. Introduction .....	137
6.1 Sampling Technique.....	138
6.2 Experimental Setup .....	139
6.2.1 Recruitment for Evaluation Exercise.....	139
6.2.2 Student Consent and Lesson Plan.....	140
6.3 Pre-assessment Skills Data Collection and Analysis.....	141
6.4 Post Evaluation and Experiential Feedback Data.....	146
6.5 Summary of Chapter .....	150
<b>Chapter 7 .....</b>	<b>151</b>
<b>Discussions .....</b>	<b>151</b>

7. Introduction .....	151
7.1 Dealing with The Research Question .....	151
7.1.1 How System Identified Gaps and Material Recommendation.....	152
7.1.2 Initial System Development Stages .....	153
7.2 Reactive System .....	154
7.2.1 Agent Long term and Short term memory .....	155
7.3 Agents Communication in The Pre-assessment System .....	155
7.4 Agent agInterface: The Interface Agent .....	159
7.4.1 Percept Observation .....	159
7.5 Agent agSupport: The Pre-assessment Agent .....	161
7.5.1 The Agent Pre-assessment Process .....	162
7.6 Strategies of the Pre-assessment System Development .....	167
7.6.1 Pre-assessment By Immediate Prerequisite Class Program Development .....	167
7.6.2 Pre-assessment By Multiple Prerequisite Classes Program Development .....	169
7.6.3 Open_Ended Answers Assessment.....	179
7.7 Agent agModelling: The Task of Classification .....	179
7.7.1 Generating Parameter Combination for Classification .....	183
7.8 Agent agModel: The Store Agent .....	185
7.9 Agent agMaterial .....	185
7.10 The Pre-assessment Sessions.....	187
7.11 Analysis of SQL Query Statements at Pre-assessment Sessions .....	187
7.11.1 Case Study I: The UPDATE Desired_Concept .....	187
7.11.2 Case Study II: The JOIN Desired_Concept.....	188
7.12 Findings from The Pre-assessment Exercise .....	191
7.13 Implications for Teaching.....	191
7.14 Relevance of Chunking in the Pre-assessment System .....	196
7.15 System's Post-Evaluation Survey .....	197
7.15.1 Student Course Distribution Data .....	197
7.15.2 User Perception of The Pre-assessment System and Sessions .....	197
7.15.3 Open-Ended User Feedback .....	198

7.16 Summary of Chapter.....	201
<b>Chapter 8 .....</b>	<b>203</b>
<b>Conclusions and Future Work.....</b>	<b>203</b>
8.1 Research Development Approach .....	203
8.2 Contributions to Knowledge .....	206
8.3 Limitation of The Study .....	207
8.3.1 Volunteer Population Sample of the Study.....	207
8.3.2 System Constraint with Jason AgentSpeak Language.....	207
8.3.3 Alternative Languages of Implementation .....	208
8.4 Further Work .....	208
8.4.1 Recommendation .....	210
<b>References .....</b>	<b>211</b>
<b>Appendix A .....</b>	<b>222</b>
A.1 Pre-assessment Data .....	222
A.2 The MySQL Tennis_Database Tables .....	231
<b>Appendix B .....</b>	<b>233</b>
B.1 Students' Feedback Questionnaire .....	233
B.2 Consent Form .....	237
B.3 Research Ethics Approval .....	238
B.4 Certificate of Volunteer Participants in the Survey.....	239

## List of Figures

Fig.1. 1: Transition State Diagram of Learning and Unlearning Processes.....	5
Fig.1. 2: Overview of The Pre-assessment System (adapted from Ehimwenma, Beer & Crowther 2015a).....	8
Fig.2. 1: Graph for RDF/XML Example: RDF resources are represented in ovals and literals in rectangles.....	14
Fig.2. 2: Comparison of RDF, RDFS and OWL languages (based on Horrocks, Patel-Schneider & Van Harmelen, 2003). .....	17
Fig.2. 3: OWL constructors and DL notation (Baader, Horrocks & Sattler, 2003). C is a class, P is a role (property), n is the number of cardinality, r is the relation.....	18
Fig.2. 4: A TBox hierarchy about family relationships. ....	18
Fig.2. 5: System prediction and motivation to achieve higher performance. ....	26
Fig.3. 1: The Structure of a Simple Reflex Agent (Russell & Norvig, 2010).....	36
Fig.3. 2: Designing Intelligent Agents: An example (Monett, 2014). ....	37
Fig.3. 3: Horizontal Architecture .....	39
Fig.3. 4: Vertical architecture: two pass.....	39
Fig.3. 5: Vertical architecture: one pass.....	39
Fig.3. 6: The Gaia model (Wooldridge, Jennings & Kinny, 2000).....	41
Fig.3. 7: The phases of the Prometheus methodology (Padgham & Winikoff, 2004) .....	44
Fig.3. 8: Major models of Prometheus (Padgham and Winikoff, 2002).....	45
Fig.3. 9: Jack code generation screen shot. The code generated are in Java,which is not the language chosen for the execution of one of the objectives of this research. ....	46
Fig.3. 10: Comparative summary of Gaia, Tropos & Prometheus. ....	47
Fig.3. 11: Components of Agent Communication Language (Dogac & Cingil, 2003) .....	51
Fig.4. 1: PDT notation symbol.....	67
Fig.4. 2: System scenario view. ....	70
Fig.4. 3: System goals specification for the pre-assessment system.....	71

Fig.4. 4: System role overview showing structured Functionalities. ....	72
Fig.4. 5: Analysis overview from system scenarios.....	73
Fig.4. 6: Agent Role Grouping.....	73
Fig.4. 7: System overview diagram. ....	74
Fig.4. 8: FIPA-compliant AUMML command protocol. ....	75
Fig.4. 9: FIPA Compliant AUMML protocol diagram analysis for inter-agent interaction. It shows the dynamic interaction of agent message passing via performatives.....	76
Fig.4. 10: AUMML Protocol Interaction table.....	77
Fig.4. 11: Detailed overview of agent <i>agInterface</i> . ....	78
Fig. 4. 12: Agent <i>agSupport</i> receiving the <i>desired_Concept</i> percept and retrieving quizzes.....	78
Fig.4. 13: Agent <i>agSupport</i> Overview: Using answer percept to make comparison. Taking pass or a fail decision, and communicating all activities and decision reached to other agents of the MAS by its agent plans. This agent also date and timestamp learning activities. ....	79
Fig.4. 14: The agent <i>agModelling</i> : The classifier agent Overview.....	79
Fig.4. 15: Agent <i>agMaterial</i> : The learning material agent Overview.....	80
Fig. 4. 16: Agent <i>agModel</i> (student) Overview .....	80
Fig.4. 17: Capability descriptor.....	81
Fig.4. 18: Expanded summary of capability descriptor: percepts, triggering events, goals, plans and data used by agents in the system.....	82
Fig.4. 19: The Pre-assessment Mechanism (Ehimwenma, Beer & Crowther (2014b) .....	87
Fig.4. 20: Strategic diagram of the Pre-assessment by immediate next prerequisite class. Where C represents the desired amongst the classes of concept and B the immediate prerequisite class to C. ....	89
Fig.4. 21: A digraph of a regular ontology tree. ....	90
Fig.4. 22: A digraph of non-regular ontology tree. A model where all the prerequisite classes under a given parent class, in this case $C_1$ , are being considered for pre-assessment.....	94
Fig.4. 23: A knowledge graph of multiple horizontal and vertical traversal .....	99

Fig.5. 1: Hierarchy of six SQL Modules Learning Structure (extended version of Ehimwenma, Beer & Crowther 2014b). .....	104
Fig.5. 2: TBox Description of an SQL Domain. ....	105
Fig.5. 3: A regular ontology of two leaf nodes per parent class node.....	109
Fig.5. 4: Linear ontological model from the TBox. SELECT is reflexive.....	110
Fig.5. 5: A non-linear hierarchy of the SQL learning structure. But some parent class nodes are not connected in sequence according to Fig. 5.1. ....	111
Fig.5. 6: A variant ontology model of the TBox description and its navigation. But not in the structured sequence presented in Fig. 5.1 .....	112
Fig.5. 7: Illustrating navigation strategy for agent !achievement goal. ....	113
Fig.5. 8: Illustrating navigation strategy based on directed links between class nodes. Yet contrasts the structured sequence in Fig. 5.1.....	113
Fig.5. 9: The insert class example with its leaf node and literal (or data) nodes. ....	114
Fig.5. 10: Jena ontology rendered in Turtle syntax.....	116
Fig.5. 11: A cross-section of the concepts: DELETE, INSERT and SELECT in structured of Figure 5.1.....	116
Fig.5. 12: Protégé OWL ontology using Turtle syntax from Jena API.....	117
Fig.5. 13: A Regular SQL ontology .....	118
Fig.5. 14: Snapshot of Agents creation and configuration in the Pre_assessment MAS Project in Jason. ....	120
Fig.5. 15: Facility of the Pre-assessment System Agent (Based on Monette, 2014)121	
Fig.5. 16: A Slice of the Java Code that gets Percept through human interaction in CArtAgO.....	123
Fig.5. 17: Snapshot of the PreassessmentGUI CArtAgO Artifact .....	124
Fig.5. 18: A slice of Jason plans that creates observable artifact and percept communication.....	124
Fig.5. 19: CArtAgO artifact for Agent Percept and User Interaction. With overlapping MAS output or display console. The output console prompts the user for inputs when the MAS is started (Ehimwenma, Beer & Crowther, 2015a). ....	125
Fig.5. 20: Agent plans based on the derived FOL syntax specified in Chapter 4 for classification of student knowledge on the DELETE desired concept. ....	127



Fig.5. 21: Inputs, communication and classification in the multiagent Pre-assessment System. Inputs are serial, as students reaction to the System. ....	129
Fig.5. 22: One vs. All Multiple Classification (Ehimwenma, Beer & Crowther, 2016a) .....	129
Fig.5. 23: A snapshot of the agent agModel (student) <i>Mind Inspection</i> of updated beliefs in Persistent beliefs after some pre-assessments by the MAS.....	130
Fig.5. 24: Agent achievement goal for retrieving and displaying the <code>deleteSelect</code> quiz from BB.....	132
Fig.5. 25: Plan snapshot for a passed answer assessment, user feedback, communication and next quiz display use of achievement goal by the agent agSupport .....	133
Fig.5. 26: Adoption of a hasKB predicate relation, and content query from BB with <code>?hasContent</code> test goal in a plan.....	134
Fig.7. 1: List of desired SQL concepts contained in a plan context and a tell Performative as means of Communication. ....	160
Fig.7. 2: Plan for Perceiving the SQL Answer Queries from the student environment. ....	161
Fig.7. 3: Adoption of the DELETE desired Concept .....	163
Fig.7. 4: Plan for a Passed Pre-assessment of <code>InsertSelect</code> .....	164
Fig.7. 5: Plan for a Failed Pre-assessment of <code>InsertSelect</code> , and giving agent the subgoal <code>!quizInsertValue(InsertValueQuiz)</code> .....	165
Fig.7. 6: Adoption of <code>!quizInsertValue</code> achievement goal, display and communication.....	166
Fig.7. 7: Initialising an iteration belief.....	168
Fig.7. 8: Testing and updating the iteration in a plan body.....	168
Fig.7. 9: Classified Decision Tree Flow for DELETE Pre-assessment .....	169
Fig.7. 10: A non-regular ontology tree.....	173
Fig.7. 11: Semantic relations of a total of 4 prerequisite leafnode of two prerequisites parent classes under <code>Join</code> .....	174
Fig.7. 12: Semantic relations of a total of 4 prerequisites leafnode for pre-assessment under the <code>Insert</code> . ....	175

Fig.7. 13: Semantic relations of a 3 prerequisite leafnodes under the Union <i>desired_Concept</i> .....	176
Fig.7. 14: Pseudo-algorithm of the pre-assessment process that depends on the number of leafnodes N considered under a <i>desired_Concept</i> .....	178
Fig.7. 15: Initialisation of iterations as beliefs in agent <i>agSupport</i> .....	178
Fig.7. 16: Two multiple prerequisite classes of 4 leafnodes classification. Agent <i>agModelling</i> sending <i>hasPrerequisite</i> predicate message. ....	181
Fig.7. 17: Two multiple prerequisite classes of 4 leafnodes classification. Agent <i>agModelling</i> sending <i>hasKB</i> predicate message. ....	182
Fig.7. 18: Classification rules generation algorithm .....	183
Fig.7. 19: Classification rules formation process.....	184
Fig.7. 20: Agent <i>agMaterial</i> use of test goal <i>?hasContent</i> before the retrieval URL materials for students . ....	186
Fig.7. 21: Percentage of number of <i>passed</i> vs. <i>failed</i> leafnode concepts.....	194
Fig.7. 22: Percentage of students' abilities. ....	194
Fig.7. 23: Time-Independent Variant Student Performance Regression Analysis based on the data in TABLE 7.6. ....	195

## List of Tables

Table 3. 1: Comparison of agent oriented programming (AOP) and platforms .....	56
Table 6. 1: Sample size of volunteers and recruitment records .....	139
Table 6. 2: Percentage of correct and incorrect pre-assessment answers .....	146
Table 6. 3: Question 1. Course of Study? .....	147
Table 6. 4: Question 2. Year of Study? .....	147
Table 6. 5: Questions 3 – 13.....	147
Table 6. 6: Question 14. What was most interesting about the session's organisation? .....	148
Table 6. 7: Question 15. What was least interesting about the session's organisation? .....	149
Table 6. 8: Question 16. What is most interesting about the SQL system?.....	149
Table 6. 9: Question 17. What was least interesting about the SQL system?.....	149
Table 7. 1: Desired_concept and order of multiple prerequisites class .....	170
Table 7. 2: The Join pre-assessment process illustration .....	175
Table 7. 3: The Insert pre-assessment process illustration.....	176
Table 7. 4: The Union pre-assessment process illustration.....	177
Table 7. 5: Summary of correct and incorrect answer responses.....	189
Table 7. 6: Time-independent variant students' performance analysis.....	195

## Abbreviations

<b>AI</b>	Artificial Intelligence
<b>ACL</b>	Agent Communication Language
<b>API</b>	Application Programming Interface
<b>AOP</b>	Agent Oriented Programming
<b>AOSE</b>	Agent Oriented Software Engineering
<b>AUML</b>	Agent Unified Modelling Language
<b>BDI</b>	Belief, Desire and Intention
<b>CARTAgO</b>	Common Artifact for Agents Open environment
<b>CBR</b>	Computer Based Reasoning
<b>DARPA</b>	Defence Advanced Research Projects Agency
<b>DBMS</b>	Database Management System
<b>DL</b>	Description Logic
<b>FIPA</b>	Foundation for Intelligent and Physical Agents
<b>FOL</b>	First Order Logic
<b>GUI</b>	Graphical User Interface
<b>HTML</b>	HyperText Markup Language
<b>ITS</b>	Intelligent Tutoring System
<b>JADE (Jade)</b>	Java Agent Development Environment Language
<b>Jason</b>	Jason AgentSpeak Programming Language
<b>KB</b>	Knowledge Base
<b>KIF</b>	Knowledge Interchange Framework
<b>KQML</b>	Knowledge Query and Manipulation Language
<b>KR</b>	Knowledge Representation
<b>MAS</b>	Multiagent Systems
<b>OWL</b>	Web Ontology Language
<b>PDT</b>	Prometheus Design Tool
<b>POMDP</b>	Partially Observable Markov Decision Process
<b>RDF</b>	Resource Description Framework
<b>SQL</b>	Structured Query Language
<b>STEM</b>	Science Technology Engineering & Mathematics
<b>URI</b>	Universal Resource Identifier

<b>URL</b>	Universal Resource Locator
<b>W3C</b>	World Wide Web Consortium
<b>XML</b>	eXtended Markup Languag

## Glossary

**Atomic formula:** This is a formula of the form  $p(t_1, \dots, t_n)$ . For example, the expression  $p(a, b)$  is an *atom* or *atomic formula* where  $a$  and  $b$  are terms or literals, and  $p$  predicate.

**Base symbol in DL:** Are primitive concepts that only occur on the right-hand-side of axioms.

**Body of a Plan:** is the course of action to be used to handle events if the plan *contexts* (or pre-conditions) are believed true at the time an agent plan is chosen to handle an event.

**Classification:** Classification in the pre-assessment system is the act by which an agent applies a set of pre-conditions in its plan *context* to match belief updates so as to categorise a student and trigger the release of learning materials, for either a pass or a fail pre-assessment.

**Context:** Represents the circumstances or conditions in which a plan can be selected for execution. They are constraints that are expected to be true before the action in a plan.

**Curriculum:** This refers to the knowledge and skills students are expected to learn. They are specific course or lessons taught by a teacher in a school.

**Desired\_Concept:** This is any of the class node concept in the SQL ontology tree that a student is expected to enter before the commencement of pre-assessment.

**Events:** Are what happens as a consequence to changes in an agent's beliefs or goals.

**Named symbol in DL:** Are the concepts being defined that occurs on the left-hand-side of axioms.

**Percepts:** Are events that are observable by agents.

**Plans:** A plan is an option of the action that an agent can select and perform. In other word, they are recipe for action or some given courses of actions. They represent agents' know-how.

**Predicate:** In logic based statements, the expression  $p(a)$  or  $p(a, b)$  is an atomic formula where  $p$  is a predicate. A predicate can be unary or binary.

**Protocols:** Are simple sequence of agents' communication using directed arrows.

**Swing:** Is a java library that provides GUI components for developing user interface.

**Triggering\_event:** Denotes the events that a plan is meant to handle.

# Chapter 1

## Introduction and Pre-Learning

### Diagnosis

#### 1. Introduction

Concepts of learning are interdependent and chronological. In human learning the successful learning of a target concept may be dependent upon relative and previously learned concepts in a given sequence of learning. Pre-learning assessment or pre-assessment as a process of learning is an enquiry into previous learning and an invitation of prerequisite knowledge into a new and higher-level concept learning. This could enhance new concept learning and improve performance. In teaching-learning environments, this process is frequently carried out by human tutors. But how can this process be replicated in an agent based system, such as, the Pre-assessment System that is designed in this study?

#### 1.1 Motivation for Study

In a learning domain, tutors teach concepts in the order of simple-to-complex or from known-to-unknown. Before a higher concept or topic is taught, lower topics in the hierarchy of learning ought to be understood. In a teaching-learning session, a tutor may probe students' prerequisite topic related to the topic that is about to be taught. In such scenarios, when the tutor asks questions, students' responses may be right or wrong. Based on this diagnosis of knowledge, the tutor is informed of the cognitive status of his students and how to begin his new teaching. Therefore, the motivation of this thesis is to investigate a strategy on an agent based system that can imitate the action of the human tutor. The system makes decisions and assembles students' knowledge status, and then recommend supplementary materials so as to close any gaps.

## 1.2 Research Question

The research problem of this work is stated in the question:

*How can students be helped to identify gaps in their current learning so that they can be fully prepared for the next stage in their learning?*

## 1.3 Purpose of The Research

The purpose of this research is to identify gaps in students' learning via a pre-learning or pre-assessment strategy, and develop a conceptual ontology to apply in the pre-assessment process on a multiagent system platform. Before the commencement of learning, students are first and foremost pre-assessed on the relative prerequisite concepts to a *desired concept*: where the *desired concept* is the intended and chosen concept of learning. This is to ascertain *strengths* or *weaknesses*, whether students possess the background knowledge to proceed to learn the chosen concept successfully.

## 1.4 Aim of The Study

The aim is to develop a model of Pre-assessment System that can pre-assess students' learning in a given domain and to use logic based rules in specifying the classification of skills and recommendation of suitable learning materials for students.

## 1.5 Objectives of The Study

The objectives of this study are as follows:

1. To investigate a systematic way of identifying gaps in students' knowledge which may hinder them in their next stage of learning. This is to allow students to self-diagnose any gaps on their previous learning before the start of a new module.
2. To build a domain ontology of related concepts and use declarative logic based representation in the system in the process of learning gap identification prior to the start of a higher and desired learning by students.



3. To investigate the communication of ontological concepts in the system in the process of identifying gaps in students' learning.
4. To develop the tools that allow the system to recommend supplementary study materials to close the gaps in their current learning.
5. To evaluate the effectiveness of the system by assessing how effective it is in helping real students improve their learning.

## 1.6 Defining The Pre-assessment System

The Pre-assessment System is an agent based elearning system that perceive the knowledge of students, communicate such knowledge, make decisions, categorise students according to knowledge assembled, and finally recommend suitable learning materials. This aforementioned processes are functionalities that are handled by a group of agents.

The domain content of the system is Structured Query Language (SQL). The system uses the example of SQL learning structure from the *Introduction to SQL* (Lans, 2006). The concepts of learning are interdependent on each other and shall be arranged in an *ontology* tree structure that is modelled after the SQL teaching materials that were made available for this work by database tutors in Sheffield Hallam University. The system keeps activities of students' during the course of pre-assessment. This is for the tutor's view so as to provide optimal assistance to students that may be facing difficulties in their SQL query constructs. In this research, the problem is a classification of students' learning activity for learning materials recommendation.

## 1.7 What is Learning?

Learning can be categorised as a change in the mental state of humans or machines after a sequence of acquired experiences. But whether these experiences have caused any changes in the '*knower*' is normally determined by some form of assessment. Inclusively, learning is search and find, recognising, classifying, grouping, separating, sorting, drawing similarities, taking instruction, or making prediction using existing knowledge. Learning is a display of intelligence which comprises information

gathering, fault detection, diagnosis and prognosis. Bratko (2001) describes learning as having to recognise a concept: If  $C$  is a concept, to learn the concept  $C$  means to learn to recognise objects [*or features*] in  $C$ . In artificial intelligence (AI), a concept is a class or object.

Learning can be permanent or temporary — meaning that a concept or process can be learned or unlearned. In a teaching-learning process, one way to determine the occurrence of learning is through some form of assessment: To ascertain whether a concept is learned or has been unlearned. In this work, the process is dichotomous, and comprises of:

- Classification of students' learning.
- Student Learning.

### 1.7.1 Classification of Students' Learning

In this work, classification refers to the selective decision making and grouping of students' responses to the quizzes, based on the desired concept entered by a student. Classification is the ability of the agent based system to recognise and classify features according to its given rules (or plans) where agents have their knowledge or beliefs represented in logic based structure. At the match of some beliefs (whether initial beliefs or update beliefs), messages are communicated interchangeably and a trigger for classification is performed to fulfill the overall goal of the agent based system.

### 1.7.2 Human Learning

Assessment is a critical catalyst for student learning (Conole & Warburton, 2005), and this is used to measure the outcome of learning. At any given stage in a learning process, this is imperative because of the need to improve students' performance. As such, assessment can be administered through one or a combination of the test techniques:

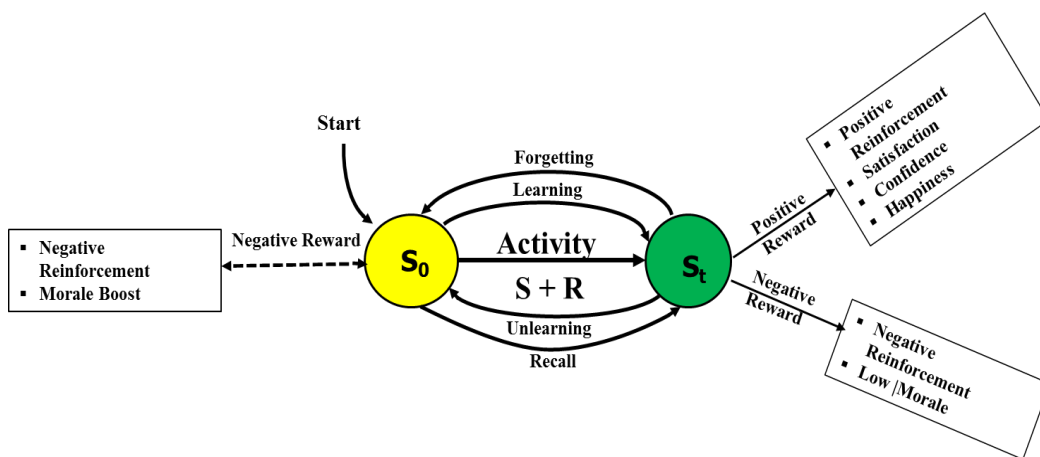
*summative* -- for grading purposes at the end of study term;

*formative* -- for immediate feedback during course of learning;

*diagnostic* – for evaluating students' prior knowledge;

*self-assessment* – for students’ reflection of own experiences and understanding. (O’Reilly & Morgan, 1999; Bull & McKenna, 2004, Conole & Warburton, 2005)

Using a schematic diagram, Figure 1.1 can be used to depict the processes of learning, unlearning and forgetting under some hypothetical activity represented as *stimulus* ( $S$ ) (e.g. question) and *response* ( $R$ ) (e.g. answer) *activity*. The Figure 1.1 maps learning, unlearning and relearning processes to some states  $S_0$  and  $S_t$ , and possible reward factors that influences learning.



**Fig.1.1:** Transition State Diagram of Learning and Unlearning Processes.

$S_0$  = Initial state (i.e. a start or previous state).

$S_t$  = Transition state (i.e. new learning state) where  $t = 1, 2, 3, \dots, n$ .

Particularly for humans, the schematic representation shows the transition states in metacognitive activities from initial state  $s_0$  to a new learning state  $s_t$  and vice versa coupled with the effect of rewards — positive or negative. This is a view from the studies of classical conditioning (Pavlov, 1960) and operant conditioning (Skinner, 1938) where positive and negative rewards were shown to influence learning.

To determine the occurrence of learning, one process to employ is the use of pre-learning diagnosis. This is vital and effective in assessing students whether the foundation is already laid for higher concept learning. In that view, skills diagnosis provides the opportunity for a pre-learning assessment of a learner’s state of knowing with regard to a given target concept. Tutors in contemporary classroom practice make

enquiries into students' prior knowledge before teaching some relatively higher concepts. This is to determine the background knowledge readiness for the new concept. When teachers give students the opportunity to explore their prior knowledge and beliefs, and then thoughtfully look and listen at what is revealed; they are gathering information for responsive instruction. This style of teaching intentionally connects what students already know with the desired outcomes (STEM, 2013).

With intelligent learning systems, students themselves can embark on self-diagnosis without the tutor's intervention in their own time, space and comfort before proceeding on the learning ladder. But most e-learning systems still do not use effective strategies for evaluating students' existing knowledge before teaching a new concept. Since knowledge is building blocks that are sequentially planned from known-to-unknown, the existence of gaps or *zone of proximal development* (Vygotsky, 1978) would inhibit the successful learning of further concept(s).

## 1.8 Need for Pre-assessment in Learning

Pre-assessment is the inquiry into relevant pre-existing knowledge at the start of a learning process to identify whether a student has the necessary background to enable them to move forward with the new material that they wish to learn. Thus pre-learning assessment creates a synergy between previous learning and the start of new learning. In the process of inquiry, pre-assessment prompts related prior learning. In the views of Conole & Warburton (2005) *diagnostic* assessment is used by tutors to determine students' prior knowledge. Andronico *et al.* (2003) state that diagnostics begins before a course of learning with the purpose of identifying what learning resources are needed by students. This is quite different from other forms of assessment. For example, *formative* assessment that is designed to provide students with feedback on progress and development whether the student understands the current teaching. Or *summative* that is used to identify the students approximate level and giving the right score or grades (Conole & Warburton, 2005; Andronico *et al.* 2003). By deduction, pre-assessment leads to better formative assessment leading to the best summative evaluation.

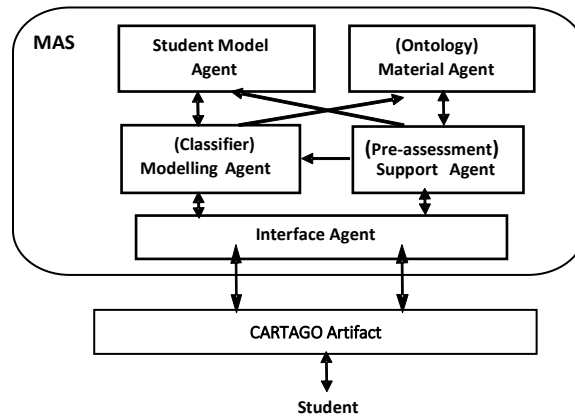
As the tutor in a face-to-face classroom context may perform a pre-learning or diagnostic assessment concerning a particular knowledge concept before teaching a higher level concept, so should intelligent tutoring systems (ITS) be modelled to assist a learner. In a virtual learning environment, one of the major problems in deploying materials for learning is ensuring that students have sufficient prior knowledge at the start of a new study session. This is made more complicated by the range of different routes that they may have taken to reach this point in their study.

Our effective approach to remedy this situation is self-assessment or self-diagnosis on prerequisite concepts to the higher concept that is desired. This way, gaps that may inhibit further knowledge may be detected and appropriate recommendation made to fill any gaps by intelligent learning systems. In so doing, students will have greater preparedness for higher or desired learning activities.

Thus this research demonstrates a pre-assessment procedure in a multiagent system (MAS) that can identify gaps in learning. The chosen tool for developing the multiagent *Pre-assessment System* is Jason AgentSpeak Language (Bordini, Hübner & Wooldridge, 2007). This is due to the language support for: belief structure in logic based representation, inter-agent communication via speech acts performatives, and persistent beliefs.

The domain content of the pre-assessment system is the SQL database. The database which is called the TENNIS\_DATABASE was modelled and hosted on the MySQL server. SQL quizzes and queries are dependent on this database, and students shall have access to the database in order to provide answers to the pre-assessment quizzes. The TENNIS\_DATABASE is made up of five data tables.

The Figure 1.2 presents an overview of the pre-assessment system and the interaction amongst the agent components. The system interacts with the user through the CArAgO (Common ARTifact for Agent Open environment) artifact. The CArAgO is the artifact (Ricci, Piunti, & Viroli; 2011) in which the multiagent system observes its input or percepts.



**Fig.1. 3:** Overview of The Pre-assessment System (adapted from Ehimwenma, Beer & Crowther 2015a)

All composite agents have their individualised tasks in their *Condition-Action* rules otherwise known as plans. These plans constitute various agent functions as designated duties within the MAS. The agents are cooperative through knowledge communication so as to achieve the overall design goal of pre-assessment, which is, to identify learning gaps in students' learning and make recommendation for learning materials via universal resource locator (URL) links. Thus the strategic purpose and functions of the Pre-assessment System are:

- 1) Perceive events.
- 2) Communicate messages via performatives.
- 3) Process perceived events (e.g. SQL concepts, query statements, logic based statement), feedback to the student, and carry out pre-assessment.
- 4) Assemble updated beliefs, match the plan that satisfies the given set of updated beliefs from an array of agent plans, and trigger classification.
- 5) While doing 4) above, dynamically keep students' activity-history for the course tutor access to unravel the technical difficulties confronting his students.
- 6) Make suitable recommendation for learning materials.

## 1.9 Contribution to Knowledge

The findings and significant contributions of this research study are:

1. Identifying gaps in students' learning using a devised Pre-assessment Mechanism.

2. Goal specification for agents using Agent oriented software engineering methodology for developing e-learning system.
3. Use of description logic syntax for defining an ontology of a learning domain.
4. Modelling classification features with logic based representation for agents for the prediction of appropriate knowledge-level learning materials.

## 1.10 Overview of Thesis

This thesis has been structured into eight Chapters. *Chapter 2* explores the literature of knowledge representation; description logic (DL) language, DL notation and symbols for knowledge modelling. This include the TBox and ABox components. The Chapter also present intelligent tutoring systems, assessment systems and multi-agents. *Chapter 3* continues with the literature on agents, agent properties, architectures and methodologies. In furtherance, the chapter discusses speech acts theory as a protocol for knowledge sharing in agent based systems, agent communication and agent oriented programming. In *Chapter 4* the conceptual development of the *Pre-assessment System* is presented using the *Prometheus* methodology. This is followed by a devised *Pre-assessment Mechanism* for the pre-assessment process, the *Student Model* parameters, and first order logic formula specification of the classifier agent reasoning process. Also discussed in the chapter is our model equation that can calculate the number of classification rules in a given ontology tree. *Chapter 5* describes the implementation of the *Pre-assessment System*. This include the various agent components, ontology models from the DL definition, and the classification procedure. In *Chapter 6* the *Pre-Assessment System* is evaluated by volunteer participants, and the data collected analysed. *Chapter 7* is discussion and explanation of findings. *Chapter 8* is conclusions and direction of further research work.

## 1.11 Publications from this Work

Elements of this work have been published and have been referenced in this thesis. Note that the terminologies and notations used in this thesis supersedes those used in the publications.

- I. Ehimwenma, K.; Beer, M. & Crowther, P. (2014a). *Ontology Engineering and Modelling for Learning Activity in a Multiagent System*. Proceedings of the 1st International Conference on Systems Informatics, Modelling and Simulation (SIMS2014), Washington, DC, IEEE Computer Society. Pp.143-147. [Ehimwenma, Beer & Crowther (2014a), Chapter 5]
- II. Ehimwenma, K. E.; Beer, M. & Crowther, P. (2014b). *Pre-assessment and Learning Recommendation Mechanism for a Multi-agent System*. In Proceedings of the 14th IEEE International Conference on Advanced Learning Technologies (ICALT 2014). IEEE Computer Society. Pp. 122-123. [Ehimwenma, Beer & Crowther (2014b), Chapters 1, 4]
- III. Ehimwenma, K. E., Beer, M., & Crowther, P. (2015a). *Adaptive Multiagent System for Learning Gap Identification Through Semantic Communication and Classified Rules Learning*. 7th International Conference on Computer Supported Education, In Doctoral Consortium (CSEDU). SCITEPRESS. Pp. 33-38. [Ehimwenma, Beer & Crowther (2015a), Chapter 1, 4, 5]
- IV. Ehimwenma, K. E., Beer, M., & Crowther, P. (2015b). *Student Modelling and Classification Rules Learning for Educational Resource Prediction in a Multiagent System*. 7th Computer Science and Electronic Engineering Conference (CEEC2015), IEEE. Pp. 59-64. [Ehimwenma, Beer & Crowther (2015b), Chapter 4]
- V. Ehimwenma, K. E., Beer, M. & Crowther, P. (Feb. 2016a). *Computational Estimate Visualisation and Evaluation of Agent Classified Rules Learning System*. International Journal of Emerging Technologies in Learning (IJET), Vol 11 (1). Pp. 38-47 [Invited paper]. [Ehimwenma, Beer & Crowther (2015a), Chapter 5]
- VI. Ehimwenma, K. E., Crowther, P. & Beer, M. (2016b). *A System of Serial Computation For Classified Rules Prediction In Non-Regular Ontology Trees*. International Journal of Artificial Intelligence and Applications (IJAI), 7(2), pp.21-33. [Ehimwenma, Crowther & Beer (2016b), Chapter 4]



# Chapter 2

## Knowledge Representation and Intelligent Tutoring Systems

### 2. Introduction

This chapter presents the background literature of description logics (DL) and knowledge representation (KR). It deals with the various forms of KR and DL support for ontology languages and development. This includes DAML + OIL, RDF(S), and OWL. The chapter describes the unary predicate, and binary predicate relation as triples in RDF and its Prolog-like ground facts equivalence for representing knowledge in a system. This herald a DL language into a TBox and its ABox counterpart, and the *condition-action* rule for symbolising a classification process for programming. The chapter also looks at intelligent tutoring systems (ITS) architectures, ITS and their strategies for supported learning. This covers multiagents in the development of ITS and analysis of some student models. The chapter also looks at some SQL assessment systems, and *Chunking*: an educational learning theory for supporting effective learning in a challenging educational environment and why it is important in this study.

### 2.1 Knowledge Representation and Ontology

An ontology is a description of things and their relationships. It represents knowledge organisation. Ontologies define objects, properties and the relationships that exists between objects (Gruber 1993; 1995), and information about an object itself (Horrocks, Patel-Schneider & Van Harmelen, 2003) in a given domain of interest.

Ontologies specifies the classes of objects that exist, the relationships amongst those classes, the possible relationships amongst instances of the classes, and constraints over those instances (Gruber 1993; 1995). In formal concepts, Maedche & Staab (2001) defined ontology as a 5-tuple  $O = \langle C; R; F; A; I \rangle$  where:

C: finite set of named **concepts** organisation.

R: finite set of binary **relations** among concepts.

F: **functions** that relates concept and relations

A: set of **axioms** that are valid in the conceptualisation.

I: set of **individuals** belonging to a domain.

## 2.2 Description Logic and Ontology Languages

Description Logic (DL) is a family of formal description languages for the representation of concepts (or classes) and their roles (known as properties or relationships) and literals (also known as individuals). Different formalisms or data structures exists for the representation of ontologies, and examples of these are OIL, OIL + DAML, RDF, OWL and answer set prolog. As a way of defining knowledge for systems, Baader, Horrocks & Sattler (2007) states that DL are the basis for ontology languages such as OIL, DAML + OIL and OWL for knowledge representation. In the following section, the various forms of knowledge representation models are presented.

### 2.2.1 SHOE: Simple HTML Ontology Extension

Frame-based languages or systems were first developed in the mid-1970s. Frame describes *Classes*, and a set of *Slots* in which slots may consist of *property-value* pairs, or a *constraint* on the value (i.e. an individual or data value). Frame was subsequently adopted by SHOE: a frame-based language with XML syntax. SHOE then became one of the earliest attempts at defining an ontology language for the web. SHOE used URI (Universal Resource Identifier) references for names that became the convention in both DAML-ONT and DAML+OIL languages (Horrocks, Patel-Schneider & Van Harmelen, 2003). SHOE was not based on RDF, and as such had lesser influence on the syntactic and semantic design of OWL.

### 2.2.2 DAML-ONT: DARPA Agent Markup Language-ONTology

The DARPA Agent Markup Language (DAML) was initiated in the year 2000 with the goal to develop a language and tool to enable the realisation of the Semantic Web

(DAML, 2006). The semantic web is the idea to represent basic fact, information or data (e.g. in document) and connect them together on the web. It is different from the connectivity of document of the hyperlink technology.

RDFS, a language that was already adopted by the World Wide Web Consortium (W3C) was to be the starting point, but lacked the much needed power of expressiveness for knowledge representation. This led to the development of DAML-ONT that extended RDF with language constructors from object-oriented and frame-based knowledge representation languages (Horrocks, Patel-Schneider & Van Harmelen, 2003). DAML-ONT was tightly integrated with RDFS. But DAML-ONT, like RDFS, was not without semantic specification issues. With DAML-ONT, it was realised that there could be disagreements, in the precise meaning of terms, both amongst human and machines in a DAML-ONT ontology.

### **2.2.3 OIL: Ontology Inference Layer**

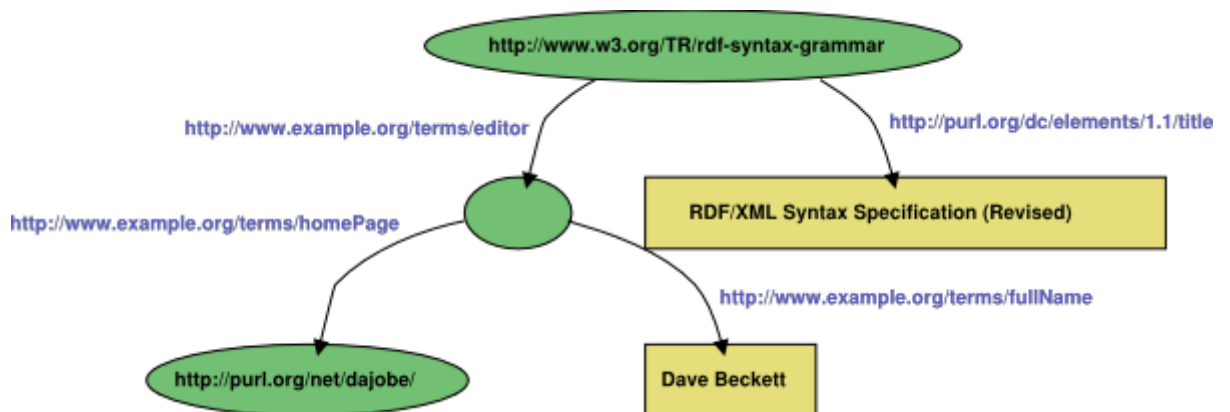
OIL is one of the languages in which OWL (Web ontology language) is based. At around the same time that DAML-ONT was developed, a group of researchers from Europe had designed the OIL language. OIL became the first ontology language to combine elements from Description Logics, frame languages and web standards such as XML and RDF (Horrocks, Patel-Schneider & Van Harmelen, 2003).

### **2.2.4 DAML+OIL**

The merger of DAML-ONT and OIL efforts produced DAML+OIL. Though, heavily influenced by OIL, DAML+OIL received additional influence from DAML-ONT and RDFS. DAML+OIL adopted a Description logic (DL) style axiom and retained and used the DL language constructors developed in OIL. But not the *frame* structure that could easily integrate with RDF syntax. Nonetheless, DAML+OIL, provided a meaning for those parts of RDF which were consistent with its own syntax and DL style model theory (Horrocks, Patel-Schneider & Van Harmelen, 2003).

### 2.2.5 RDF: Resource Description Framework

RDF is a graph database. It is a standard model for data interchange on the Web (W3C, 2014). RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (known as “triple”) (Fig.2.1). This linking structure forms a directed, labelled graph, where the edges represent the named link between two resources, represented by the graph nodes (W3C, 2014). RDF are triples  $(a, P, b)$  or set of triples which are expressed as *logical formulas*  $P(a, b)$ : This is a binary statement in which the binary **predicate**  $P$  relates the subject  $a$  to object  $b$ . RDF are binary predicates only. The relationships or graphical connectedness between a node *subject*  $a$  and a node *object*  $b$  via a predicate  $P$  is a semantic net. RDF has been given the syntax of XML (W3C, 2004). RDF is very scalable, but is not very expressive and does not provide support for semantics (W3C, 2004). RDF is not data format, but a data model with a choice of syntaxes for storing data (DuCharme, 2013).



**Fig.2. 1:** Graph for RDF/XML Example: RDF resources are represented in ovals and literals in rectangles.

**Source:** <https://www.w3.org/TR/REC-rdf-syntax/>

The edges (arrow-head lines) go from a resource to any other resource or to a literal, and never from a literal to a resource or another literal. So in RDF representation, literals are the terminal values of a resource. Simply put, RDF resources and edges are URIs, literals are not, but simply values e.g. universal resource locator (URL).

All web URLs are URIs but not all URIs are URLs.

Thus RDF vocabulary is the set of URIs for the edges that make up the RDF graphs—so the use of common URIs is synonymous to act of communicating in an understandable language—hence the term vocabulary. For two semantic webs to share data there needs to exist a common vocabulary or keyword. Similarly, the model of agent communication in FIPA is also based on this assumption that two agents, who wish to converse, must share a common knowledge of the ontology for the domain of discourse. That is the agents must ascribe the same meaning to the symbols used in the message (FIPA, 2000).

### **2.2.6 RDFS : Resource Description Framework Schema**

RDFS is expressed as RDF. RDFS is *object oriented* in its nature. That is, it is fundamentally about describing classes of objects. Its supports semantics of data by class and properties descriptions, class hierarchies and inheritance, and property hierarchy. RDFS gives flexibility to the definition of data in that a data of a particular class may be expressed to have various type declaration i.e. ***RDFS:type*** or different property declaration i.e. ***RDFS:property***.

### **2.2.7 OWL**

The development of OWL has been influenced by several ontology languages. For example, RDFS, SHOE, OIL, DAML-ONT and DAML+OIL. But DAML+OIL has heavily influenced the emergence of OWL (Horrocks, Patel-Schneider & Van Harmelen, 2003). OWL is an increasingly expressive language. For example, one of such expressiveness is its power to specify property values and validate relationships while maintaining upward compatibility with RDF and RDFS. OWL has three sublanguages, which are Owl Lite, OWL DL and OWL Full.

- **Owl Lite**

OWL Lite is termed as the *simpler* OWL DL expression language. The language is based on the SHIF(D) version of description logic language which allows complex class descriptions, specification of conjunction, disjunction, negation, existential and universal value restrictions, role hierarchies, transitive roles, inverse roles and restricted form of cardinality constraints (cardinality 0 or 1) and support for concrete

domains (Horrocks, Patel-Schneider & Van Harmelen, 2003, de Bruijn *et al.* 2004). Its support for constraint features are simple (Laclavik *et al.* 2012).

- **OWL DL**

This is the SHOIN(D) variant of description logic language (Horrocks and Patel-Schneider, 2003; de Bruijn *et al.* 2004). OWL DL is more expressive than OWL Lite. It provides additional support for individual names in class descriptions (also called nominals) and allow arbitrary cardinality restrictions (de Bruijn *et al.* 2004). OWL DL is equivalent to DAML + OIL. OWL DL constructs are with restrictions such as:

- a class cannot be both an individual (instances) and property
- a property cannot be an individual as well as a class (Laclavik *et al.* (2012).

- **OWL Full**

OWL Full gives greater freedom for expressiveness by allowing the syntax and semantics use of both OWL DL and RDFS languages (de Bruijn *et al.* 2004). For example, while a class cannot be both individual and property in OWL DL as stated above; in OWL Full, a class can be both. OWL Full is not restricted to DL, and it is also very close to first-order logic (FOL).

In the Fig. 2.2 a comparison and the relationship between RDF, RDFS and OWL languages is given. There are different approaches for building the agent knowledge model, but the internal knowledge model of agents is left for an agent programmer (Laclavik *et al.* 2012).

## 2.3 TBox Terminology

Knowledge representation system based on DLs consists of two components - TBox and ABox (Obitko, 2007). TBox is a knowledge representation (KR) formalism that represents the knowledge of an application domain (the world) by defining relevant concepts (expressions) in that domain and then using these concepts to specify properties of individuals occurring in the domain (the world description). Nardi and Brachman (2003) state that TBox contains *intensional* knowledge in the form of a terminology or taxonomy and is built through declarations that describe general properties of concepts. The “terminology” denotes hierarchical structure built to

provide an intensional representation of the domain of interest (Nardi and Brachman, 2003).

RDF	RDFS	OWL
<ul style="list-style-type: none"> <li>* Domain independent.</li> <li>* States fact in triple and establishing the relation between two ends.</li> </ul>	<ul style="list-style-type: none"> <li>* provide mechanism for defining specific domain.</li> <li>* States class and property relation.</li> <li>* Declares class and subclasses in subsumption hierarchy, supports property and subproperty, domain and range restriction.</li> <li>* Logical combinations beyond its use.</li> </ul>	<ul style="list-style-type: none"> <li>* Compatible with several existing ontology languages e.g. OIL, DAML + OIL.</li> <li>* Extends RDF fact stating ability, and RDFS class and property structure ability.</li> <li>* Declares class and subclasses in subsumption hierarchy</li> <li>* Classes can be logical combinations (intersection, union, negation) of other classes. Or as enumeration of other specified object.</li> <li>* Extends RDFS by declaring properties as transitive, symmetric, functional or inverse.</li> <li>* Expresses disjoint, equivalence, individuality of object, quantification and value restriction.</li> </ul>

**Fig.2. 2:** Comparison of RDF, RDFS and OWL languages (based on Horrocks, Patel-Schneider & Van Harmelen, 2003).

A DL system is a combination of a TBox and ABox. The term ABox and TBox which are used to describe two-different but-related kinds of statements for ontologies together make up a knowledge base. The Figure 2.3 is a table showing the DL syntax notations for expressing logical axioms or statements in DL. A TBox describes the vocabulary or the classes of objects that make up a KB in an application domain. Basically this vocabulary are the concepts (set of individuals) plus the roles (relationship between concepts). The Figure 2.4 is a TBox description of some modelled axioms in a family domain (Baader & Nutt, 2003). The left hand side of the equality sign is where the *named symbol* (defined concepts) known as the atomic

concept occurs, and on the right hand side is the *base symbol* also known as the primitive concepts.

Constructor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \dots \sqcap C_n$	Human $\sqcap$ Male
unionOf	$C_1 \sqcup \dots \sqcup C_n$	Doctor $\sqcup$ Lawyer
complementOf	$\neg C$	$\neg$ Male
oneOf	$\{x_1 \dots x_n\}$	{john, mary}
allValuesFrom	$\forall P.C$	$\forall$ hasChild.Doctor
someValuesFrom	$\exists r.C$	$\exists$ hasChild.Lawyer
hasValue	$\exists r.\{x\}$	$\exists$ citizenOf.{USA}
minCardinality	$(\geq n r)$	$(\geq 2$ hasChild)
maxCardinality	$(\leq n r)$	$(\leq 1$ hasChild)
inverseOf	$r^-$	hasChild <sup>-</sup>

Fig.2. 3: OWL constructors and DL notation (Baader, Horrocks & Sattler, 2003). C is a class, P is a role (property), n is the number of cardinality, r is the relation.

Woman	$\equiv$	Person $\sqcap$ Female
Man	$\equiv$	Person $\sqcap \neg$ Woman
Mother	$\equiv$	Woman $\sqcap \exists$ hasChild.Person
Father	$\equiv$	Man $\sqcap \exists$ hasChild.Person
Parent	$\equiv$	Father $\sqcup$ Mother
Grandmother	$\equiv$	Mother $\sqcap \exists$ hasChild.Parent
MotherWithManyChildren	$\equiv$	Mother $\sqcap \geq 3$ hasChild
MotherWithoutDaughter	$\equiv$	Mother $\sqcap \forall$ hasChild. $\neg$ Woman
Wife	$\equiv$	Woman $\sqcap \exists$ hasHusband.Man

Fig.2. 4: A TBox hierarchy about family relationships.

From the TBox terminology in Figure 2.4, the axiom

$$\text{Human} \sqcap \neg \text{Female} \sqcap (\exists \text{married.Doctor}) \sqcap (\forall \text{hasChild.}(\text{Doctor} \sqcup \text{Professor}))$$

then defines the concept of “A man that is married to a doctor, and all of whose children are either doctors or professors” (Baader, Horrocks & Sattler, 2003).



## 2.4 ABox World Description

The term ABox (*Assertion Box*) which complements the TBox are assertions about named individuals in terms of the vocabulary described in a TBox. Precisely, the ABox contains assertional knowledge called *ground fact* (Rudolph, 2011) which is a description of world. It asserts and introduces named individuals of the world, and their properties. Properties can be unary and binary. A *unary property* specifies what class a named individual belongs while the *binary property* specifies the relationships also known as *role* between two named individuals. Given that  $C$  is an atomic concept,  $R$  as role concept, and  $a$ ,  $b$ , and  $c$  as individuals, it follows that (Baader & Nutts, 2003; Rudolph, 2011):

1.  $C(a)$  – concept assertions implies  $a$  belongs to  $C$ ,
2.  $R(b, c)$  – role assertions implies  $c$  is a filler of the *role*  $R$  for  $b$ .

According to Baader & Nutts (2003), if *Peter*, *Paul* and *Mary* are individuals, the following are constituents of an ABox assertions from the TBox in Figure 2.4:

*MotherWithoutDaughter(mary)*  
*Father(peter)*  
*hasChild(mary, peter)*  
*hasChild(peter, harry)*  
*hasChild(mary, paul)*

## 2.5 Answer Sets Prolog

Answer Set Programming or Prolog (ASP) is a language for knowledge representation and reasoning based on the answer set logic programs (Gelfond, 2008; Baral & Gelfond, 1994). ASP or language allows domain and problem-specific knowledge, including incomplete knowledge, defaults, and preferences, to be represented in an intuitive and natural way (Brewka, Eiter, & Truszczyński, 2011). ASP is an approach to declarative programming whereby in a declarative style, a problem or the world description are specified declaratively. ASP has its roots in deductive databases, logic

programming, logic based knowledge representation and reasoning, constraint [rules] solving, and satisfiability testing (Hölldobler & Schweizer, 2014).

A logic program is a set of rules of form, and ASP models are declarative and consist of rules likened to those in Prolog (Gelfond, & Lifschitz, 1988; Lifschitz, 2008) such as:

$$A \leftarrow L_1, \dots, L_m$$

where  $A$  is an *atom* and *head* of the rule, and  $L_1, \dots, L_m$  are *literals* and *body* of rule.

Thus

$$\begin{aligned} & p(1), \\ & q(2), \\ & q(x) \leftarrow p(x). \end{aligned}$$

can be a model of a program.

More so,

$$\begin{aligned} & q(a, 1). \\ & q(b, 2). \\ & p(X) \leftarrow K + 1 < 2, \\ & q(X, K). \\ & r(X) \leftarrow \text{not } p(X). \end{aligned}$$

is a program of Answer Set Prolog containing two *facts*, and two *rules*, where  $p$ ,  $q$ , and  $r$  are *predicates*; and  $X$  and  $K$  are variables. A *program* is called *ground* if its *terms*, *literals* and *rules* are *ground*. That is, if the program contains no variable and no symbol for arithmetic function (Gelfond, 2008). A fact being *ground* is contained and used in the program.

In the description of knowledge bases (KB), answer set models as a knowledge representation language can be combined with description logic to represent facts and to reason about facts. This is a situation where ABox and Answer Set program models draw on some similarities. In Gelfond (2008), a basic methodology for representing knowledge was described using open-ended signatures which are names, courses, and departments to constitute some KB facts (a collection of departmental record):

$$\begin{aligned} & \text{member}(\text{sam}, \text{cs}). \\ & \text{member}(\text{bob}, \text{cs}). \\ & \text{teaches}(\text{sam}, \text{cs}). \end{aligned}$$

*course(java, cs).*  
*course(c, cs).*  
*course(ai, cs).*  
*course(logic, cs).*

together with the closed-world assumptions expressed by the rules:

*teaches(P, C) ← member(P, cs),*  
*Course(C, cs),*  
*teaches(P, C).*

Which states that

*if the variable P is a member of cs,*  
*and the variable C is a cs Course,*  
*and the variable P does teach C;*  
**then** *conclude that the variable P that matches sam*  
*teaches a Course in cs.*

Thus, *teaches(sam, cs)* is returned because the conditions which are contained in the ground facts are satisfied in the program. Like ABox, ASP allows the expression of KR in both both unary and binary form. This form of KR formalism that constitute atoms (or constants) have also been expressed in prolog-like rules for program execution, for example (Eiter et al. 2008, p.1501; Zini & Sterling, 1999; Brewka, Eiter, & Truszczyński, 2011).

In Zini and Sterling (1999) for instance, the knowledge represented was for multiagent system that comprised of four agents. The KB which are a representation of a *Sports ontology* (Zini and Sterling 1999) were specified as follows:

*sport(cycling)*  
*sport(soccer)*

which are *unary* declaration stating that *cycling* and *soccer* are types of *sports*; and

*competition\_of(seriea; soccer)*

a *binary* declaration which states that *seriea* is a league *competition of soccer*. Wu, Zeng & Yang (2008) state that in DLs, the conceptual knowledge of an application domain is represented in terms of *concepts* (unary predicates) that are interpreted as sets of individuals, and *roles* (binary predicates) that are interpreted as binary relations

between individuals. Thus, in the *Sports* ontology, the unary predicate *sport* is property of both *cycling* and *soccer*, respectively, while the binary predicate *competition\_of* is a relation between *seriea* and *soccer* literals.

## **2.6 Classification**

Classification is *feature, instance* or *attribute* learning. It is when features (inputs or training set) that are symbolised in a system have corresponding class labels (i.e. outputs) to predict. These features can be continuous, categorical or boolean (Kotsiantis, Zaharakis & Pintelas, 2007). Classification consists of taking input vectors or data and deciding which *N* classes they belong to after running them through a classifier(s) (Rifkin & Klautau, 2004; Marsland, 2014). While most classification system is the support vector machine, this thesis considers an agent based classifier for students' learning.

Having looked at the various ontology languages for representing knowledge for systems, the act of classification in this research is not about the grouping of nodes in an ontology tree. But the collection of information about the knowledge status of students and the recommendation of the appropriate or a set of appropriate learning materials based on the available information to the system. The decision process in which students are categorised is through *condition-action* rules.

## **2.7 Condition-Action Rule**

In a classification system, decision rules are the fundamental knowledge that are compared and matched with available information or known facts, and subsequently utilised by the system to perform the act of classification or conclusions. Rules of this nature have two component parts: the left-hand side known as the antecedent, condition, premise or situation, and the right-hand side part referred to as the consequent, action, conclusions, response, or prediction (Patterson, 1990). This is the logical structure of a rule based system where a classification system is given a reasoning task about some available knowledge or concepts in order to draw conclusions about some incoming data. In Hutchinson (1994) such methods can be used for learning *concepts*: In AI (artificial intelligence), a concept is treated as a

formal definition or predicate. For most of these systems to work, Hutchinson (1994, p.310) states that in a learning system the following assumptions are valid:

- *Conditions which are basic predicates for testing a state must be specified in advance:* This is preparing rules that must be satisfied as pre-conditions for the system or a component of the system.
- *The predicates are the essential part of the language or formalism for task representation:* All the variables in the environment should be gathered for adequate representation in the system.
- *There must be something—set of rules—to learn:* For a system to make decisions, a set of rules must be specified according to the environment and variables in the problem.
- *The training set is clean or devoid of noisy relations:* In that case, the data used for preparing the rules for the system must be unambiguous to be suitable to match the incoming unknown data or information.
- *The training set should contain counter-examples:* All examples (or facts) that may be available to a system may not be similar. Some may be positive and others negative. Rules should be stated to cover both positive and negative facts.
- *Basic predicates can be partitioned into independent group:* Different variables that are related can be grouped in one rule.
- *Within each group, the predicates are mutually exclusive and cover all cases:* No case of classification much be missed. Otherwise, this would result in the misclassification of an object.

The rule based systems are **IF** <conditions> **THEN** <actions> rules, where the set of <conditions> are needed to be matched and satisfied before the <actions> part is triggered.

## 2.8 Intelligent Tutoring and Learning Systems

Intelligent Tutoring Systems (ITS) are applications that employ AI: artificial intelligence to education and instructional design (Rossi & Fedeli, 2012), or AI techniques in computer programs to facilitate [human] learning (Padayachee, 2002).

ITS are computerised learning environments that incorporate computational models in the cognitive sciences, learning sciences, computational linguistics, artificial intelligence, mathematics, and other fields that develop intelligent systems that are well-specified computationally (Graesser, Hu & McNamara, 2005). ITSs are cognitive architectures that interact heavily with humans when supporting them in one of the hardest cognitive process i.e. learning (Pipitone, Cannella & Pirrone, 2012). Several ITS exist with support for a given level of adaptability but must be able to present material at a level of difficulty and detail suited to the state of knowledge of the student, and to do so, the system must know and follow the student's changing knowledge (Michalski, Carbonell & Mitchell, 2013). This is achieved by a set of carefully planned rules (Hutchinson (1994) where a set of outputs are provided for some given set of inputs. Integrating supervised classification technique into ITS development is aimed at making accurate class predictions that suits an individual student's need and level of knowledge.

## **2.9 SQL Assessment and Learning System**

A database is a repository of information organised in such a way that it can be accessed, managed and updated easily. A database is created, stored and maintained on a database management system (DBMS). DBMS interacts with a user, connects with other application or other databases. Examples of DBMS are MySQL, PostgreSQL and HyperSQL to mention a few.

SQL (Structured Query Language) is the dominant database language (Abelló *et al.* 2008). In Kenny & Pahl (2005) SQL is a formal declarative database programming language that comprise data manipulation keywords such as select, from, where, delete, insert, into, update, set, on, and join to mention a few. The skills in SQL are challenging and students have many difficulties learning them (Mitrovic, 1998). In the perspective of Prior (2003) learning and mastering of these skills is a difficult process that requires considerable practice and effort on the part of the student. One of the challenges is mapping a statement of problem given in natural language into the information that is required from the database in an appropriate SQL statement; this Prior (2003) stated is not easy. Another difficulty is students' misunderstanding of the

basic elements of SQL and first order logic and the relational data model in general (Dekeyser, de Raadt & Lee, 2007).

To support students with the learning of SQL and determine individual students' SQL query formulation skills, the AssesSQL (Prior, 2003; Prior, & Lister, 2004) was developed. The research examined the difficulty faced in the assessment of students' SQL query skills, and encourage students to use structured query language as SQL professions. For assessment, the system present questions to student, and expects students to enter query solution to the question. The AssesSQL query content covers only the SELECT statements.

In the LEARN-SQL tool, Abelló *et al.* (2008) implemented a strategy that objectively allows the evaluation of the correctness of the solution to a question given by a student by providing automatic correction to queries by comparing the students' solution to all existing valid solutions in the system. The system, tests, feedback and grade students in their learning of SQL. The LEARN-SQL was developed and comprised statements such as the SELECT and UPDATE queries. This is from the backdrop of previously development SQL systems whose content only covered the SELECT statements (Abelló *et al.* 2008).

There also exists a number of sites that provides tutorial to students on SQL learning. Examples are "w3schools.com/sql", "Beginner SQL Tutorial" and "SQLCourse.com" that have lists of modules from which a student can make a choice in order to start learning; and the "SQLzoo.net" that provides support through multiple choice (objective type) quizzes. While they provide ability for students to run queries or take quizzes, they do not provide assistance or recommendation for errors and requisite learning.

## **2.10 Chunking: An Educational Theory of Learning**

In learning and learning technologies, the basic goal of instruction is to ensure materials are learned and understand for the advancement of learning. But students often face difficulty in their learning. Managing skills in smaller components known

as *Chunking* has helped to facilitate effective learning (Casteel, 1988; Anderson, 2008). *Chunking* is a procedure of breaking skills, learning materials or information into smaller, more manageable units for students to succeed.

## 2.11 Approaches to Agent Based Learning and Formative Assessment Systems

In Abdullah, Malibari & Alkhozae (2014), Adaptive Boosting (AdaBoost) multiagent-based system was used to mine students' historical data to classify and predict students' progress. Based on the current data, the prediction agent would receive a communication request, and would then make a grade prediction. Experimental results obtained showed that with accurate classification, students who got low performance prediction had the reasons for this analysed by the system, and were subsequently motivated by the system to achieve high performances (Fig. 2.5).

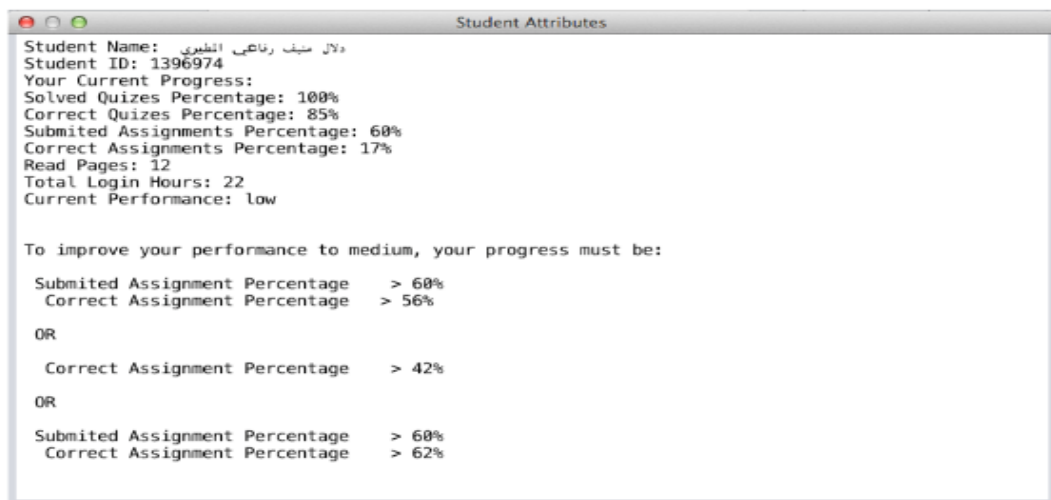


Fig.2. 5: System prediction and motivation to achieve higher performance.

In González, Burguillo & Llamas (2005) case-based reasoning approach was used to model students in a multiagent systems for learning. Case-Based Reasoning (CBR) is a problem-solving paradigm that is able to utilise the specific knowledge gained from previous experiences in similar situations (cases) to solve a new problem. At the start, a student new to the system is asked to take some tests. The system then analyses the tests results to gather information about the student. This approach categorises students



according to knowledge level and their learning preferences, however it was devoid of the assessment *question selection* strategy.

Chadli, Bendella & Tranvouez (2015) addressed how students should be evaluated using multiagent system simulation. The approach employed fuzzy set theory and agents' negotiation, and was based on an *evaluation model* that: identifies skills in the domain, student skills comparison with the background skill, and evaluation of student ability. From the experimental results, it was stated that the simulated model provided assessments similar to that of an expert and significantly improved learners' performance.

In Rosbottom & Moulin (1998) a different approach was proposed for student assessment and presentation of materials for learning in a multi-agent adaptive course delivery system on Euclidean Geometry. The approach was based on probabilistic models in which student behaviours at the interface of the system were interpreted, and prediction for the next stage of learning was made.

The application of multiagent system for educational games in learning has been reported as well. Dutchuk, Muhammadi & Lin (2009) presented work on the development of Multi-Agent System-based educational game called QuizMAster for e-learning. The game helped students learn their course material through friendly competition. Their research explored the use of perceptive pedagogical agents that would determine the learners' attitudes and emotional states by examining their: understanding, response timing, history, banter [humour]; and provide appropriate feedback to students in order to motivate them for learning.

Using two different computational intelligent techniques, Alexakos *et al.* (2006) addressed e-learning assessment on the platform of a multiagent system. The agents provided intelligent assessment services based on Bayesian Networks and Genetic Algorithms. Based on the Bayesian Networks' techniques, the system managed the questioners of an e-learning system using Bayesian Networks of probabilities that capture the probabilistic relationship between variables, as well as historical information about their relationship. From the report, results indicate that the agent platform provided assessment services.

In Wang (2014) a Partially Observable Markov Decision Process (POMDP) framework combined with reinforcement learning (RL) for building an ITS was proposed. The systems main component state comprised of: *actions*, *observations* and a *policy*. The POMDP intelligent technique was chosen on the premise that the agent cannot fully observe the knowledge state of students for it [agent] to take action. On the system, the agent partially observes students' input, and the system takes actions. To practically use the system, a student would ask a question (about a concept), the system would choose an answer and present to the student; then another question is asked, and the system would answer, and so on. The responses from the student thus determines the agent policy i.e. the teaching strategy. In this approach, the students are not assessed. The ITS teaches based on the questions asked by students. In this type of strategy, though, students' skills were not categorically measured, but the system provided support to students' learning. This is viewed in such way that, the questions asked by students are the issues bordering around their learning. Despite the assistance rendered by this ITS, a formal or formative assessment would still be required for formal qualification or higher concept learning.

Yu & Zhiping (2008) proposed intelligent pedagogical agent for evaluating prior knowledge based on the selective categorisation of learners as: *novice*, *beginner*, *intermediate*, or *advanced* learners where the learners themselves make the decision in selecting the group they think they fit-in before they start learning. Issues with this strategy is that students may misjudge the best learning category that may suit their own learning needs.

In an approach to meet learners' needs, Gamalel-Din (2002) proposed the development of the SmartTutor. As an agent based approach to support learning, SmartTutor was prescribed with two major models: student model and teacher model. The teacher model uses the concepts of Case-based reasoning for representing instructor past experience (i.e. teaching strategy & capability) where each case represents an approach for teaching a certain concept. The student model uses inductive learning-by-experience component to adapt to expected student prerequisite profile and group students together for tutors according to the different tutors teaching strategy and

capability. In SmartTutor, the instructor defines the prerequisite skills he believes the student can follow to gain new skills. While the strategy can effectively keep track of the lectures visited and content presented, SmartTutor would not identify the technical skill gaps required by students. The strategy is more tailored towards the instructors' advantage rather than the students because the identified group of students are tutored together, thereby reducing the tutors' workload.

## **2.12 Recommender Systems in Education**

Recommendation systems in adaptive learning propose and prescribe content and items that centres around the learning needs of students. This is quite different from recommender systems for buying products because learning is an effort intensive task that requires more time and interaction on the part of students compared to commercial transactions (Manouselis *et al.* 2011). Furthermore, that learners rarely achieve a final end state. Based on the fact that there are levels in learning. Instead of buying a product and owning it, learners achieve different levels of competences that have various levels in different domains. Thus in such situation, what is important is identifying the relevant learning goals and supporting learners in achieving them (p.6).

In the views of Bañeres (2017) adaptive or personalised learning tends to model learners' learning path, activities and educational resource. To this end, several e-learning recommender systems have been proposed. In Bañeres (2017) for instance, a standalone quasi-summative assessment model was proposed to boost instruction process and customisation of learning path. In the model, students are graded based on some learning activities using a model of equation, and the adaption on the students' preferences and effort spent on course. Should a learner fail an activity, it means the competence needed has not been completely acquired; and this could hinder further learning.

El Mabrouk, Gaou & Rtili (2017) also proposed a recommender system that can recommend the most appropriate content for learning. The system architecture comprises four interactive modules, namely: i) data collection part that is based on users' profiles and interest; ii) information processing unit for the learning model, user

classification and content classification; iii) recommendation module; and iv) log file component for the recommended classes meant for use in future reclassification. The system matches users' interests with content categories and classify users according to e.g. content submitted, subjects, and item ratings, respectively. Like El Mabrouk, Gaou & Rtili (2017) proposed recommender system, several classification systems employ the use of multiple components with different functions in order to fulfil the task of classification or recommendation. Thus multi-components in a recommender system draws similarity with multiagents to solve a problem. However, the aforementioned proposed system is not the kind that would assess students' skills before making recommendation. This is similar to the recommender system proposed in Bañeres & Conesa (2017) in which the system supports users to tick through a set of checkboxes such as *Completed Courses* or *Not Completed Courses* so as to classify users whether they possess the requisite skills for a given job. Though the system is geared towards employability skills classification, it could assist users in recognising their areas of skills limitation and then focus on the desirable skills. The system does not provide any form of skills assessment.

One other assessment and learning tool is the PAT Tutor (Ritter et al 1998) -- an ITS for teaching introductory algebra. In PAT, learning task and exercises are arranged in sections at different skills level as specified in a standard mathematics curriculum. When students demonstrate mastery of a section (by achieving a level of competence on all underlying skills), the Tutor system promotes the student to a new section, which includes some new skills (Ritter et al 1998). In this strategy, students' knowledge is assessed before moving to a higher level. Which means that the system can ascertain that a set of competences have been achieved before promotion to other skills.

## **2.13 Student Modelling**

Students modelling components or attributes determines the effectiveness of intelligent tutoring systems. The method used in representing the knowledge of students is referred to as the *Student Model* (Baffes, 1994). Since the 1970s, several programmed learning methods have been used in modelling the components of students in learning. Padayachee (2002) states that ITS architectures can be classified

into three categories, namely: traditional three-model, classical four-model and new-generation architectures.

### 2.13.1 Traditional Three-Model

These ITSs models comprise three major components in their design, namely:

- **Domain Model:** This is the component that contains the knowledge relating to the subject matter or content. It answers student arbitrary questions, and provide alternative explanations to the same concept.
- **Student Model:** This is the component that holds the students emerging knowledge and skills.
- **Tutoring Model:** Is the component that provide the knowledge towards the learning goals and has control over the sequence and selection of subject materials. It can diagnose misconception and learning needs.

### 2.13.2 Classical Four-Model

As well as maintaining the components of the Traditional Three-Model, an additional *User Interface* as a fourth component is added to this model. Systems of this architectural type have integrated modules named as:

- **Knowledge Base:** This component is similar to the domain model of the Three Model Architecture. In this model, the subject tutor puts together declarative knowledge (what to learn), and the procedural knowledge (how to learn) in the system.
- **Student Model:** Stores information about student knowledge and skills, and student cognitive processes. It maintains strategy that helps students to learn from errors.
- **Pedagogical Module:** This module is similar to the Tutoring component of the Three Model Architecture. This component uses the current learner's state to select an appropriate learning path to accomplish a learning goal.
- **User Interface:** This is the user interface where dialog between the system and the user are ensured.

### 2.13.3 New-Generation Architectures

A prominent model of this type of architecture are those such as proposed on the platform of multi-agent systems (MAS) for learning purposes. As modular entities that are created to form a group of cooperative components, a MAS developed. Within the systems, Padayachee (2002) states that the ITS architecture comprises an interface agent with a function to interface between the learner and system, a communication agent that ensures interaction between agent components, and a “*micro-society*” of agents that may cooperate to solve a problem activity in a formal and well-structured knowledge domain. Agents are computational entities that are modelled after the human cognitive framework. Each ITS agent or micro-society of agent have their micro-specialities or functions. To achieve the overall function of the system, agents uses structured knowledge and communicative means. This is emphasised by the social organisational perspective of the *Gaia* methodology (Wooldridge *et al.* 2000) that is presented in Chapter 3.

### 2.14 Summary of Chapter

This chapter has presented knowledge representation (KR) and various representation languages. It discussed description logic as the language that supports the development of KR languages such as OIL, DAML + OIL, RDF, RDFS, OWL, TBox, ABox and answer set prolog (ASP). The chapter analysed ASP as a KR language in unary and binary predicates. While the unary predicate is of the form  $p(a)$ , the binary predicate is the form  $p(a, b)$  which is synonymous to RDF like triple and first-order logic representation. A type of data representation form in agent based systems. Due to OWL DL power of expressiveness, in Chapter 5, the ontology of the content of learning of this thesis shall be presented in DL language.

The chapter also discussed intelligent tutoring systems (ITS), categories of student model ITS, SQL learning and assessment systems, recommender systems, and agent based systems for assessments and learning. The literature unveiled that recommended learning is an effort and it is time consuming on the part of students, and of particular interest to this thesis, SQL is not a language that is easy to learn. It is one that requires considerable effort from students to understand, and one of the significant challenges

faced by students is the interpretation of a statement of problem in natural language into its SQL equivalent query statement. Then, a few examples of SQL system were examined. Each with different strategies for evaluating students SQL queries, but with a similar process of testing students queries which involves the comparison of students' queries with the system underlying predefined answers to questions. It was gathered from literature that SQL is challenging and difficult. Then one of the educational design principles of learning known as *Chunking* was looked into. This is in view of how *Chunking* could be applied in the design of an SQL system so as to allow students pay attention to the small units of skills recommended for learning within a given assessment; and not on a long waiting lists of recommended materials to learn. This way, *Chunking* prevents fatigue, and boosts enthusiasm in learning.

The literature then surveyed some strategies that have been combined with multiagent development for supported learning. But with a few actually targeted at the misconception, misunderstanding or gaps in students' learning. For instance, in the QuizMAster system, the system supports student to learn through friendly competition. But this is only by examining the learner's attitude and emotional states. An approach that provide motivation to learning and appropriate feedback, but not content of learning. A similar approach is accounted for in the multi-agent adaptive course delivery system on Euclidean Geometry, where prediction for next stage of learning is by agents' monitoring of physical behaviour of students at the interface. This approach will certainly not gauge the appropriate material for next learning. Alexakos *et al.* (2006) and González, Burguillo & Llamas (2005) case-based reasoning approaches to support learning with the application of agent based systems assessed students for learning. But the strategy for question selection was not reported. Question selection strategy is determined by the kind of assessment being considered. The AdaBoost (Abdullah, Malibari & Alkhozai, 2014) approach used historical data to learn current data for the classification and prediction of students' grade. The system compares grades to gauge students' progress, not giving attention to the critical cognitive areas that can cause low performances. The best strategy for supporting real time learning is the identification of skills. This was addressed in Chadli, Bendella & Tranvouez (2015) by identifying domain skills in the system, comparison of students' skill and evaluation of student ability. This type of model was targeted at unravelling

the skills set of students in that domain, and would inform the tutor where the strengths and weaknesses lies. The chapter also presented three categories of student model architectures for designing intelligent tutoring systems. From Padayachee (2002) the new generation student model architecture was stated as those models that supports multiagent system development. Looking at the models, components of the Classical Four-Model architecture can be integrated into the new model architecture of multiagent systems. This involves the knowledge base which holds the target knowledge, the student module that store students' cognitive states, pedagogical module that has the teaching strategy or sequence for efficient selection of learning path; and user interface for interactive dialog. The next Chapter 3 continues with literature survey on agents and multiagents.



# Chapter 3

## Agents, Agent Oriented

## Methodologies and Interaction

### 3. Introduction

In Chapter 2, the history of different knowledge representation (KR) languages for specifying knowledge was presented as well as intelligent tutoring systems, their architectures and multiagent systems for educational purposes. This Chapter 3 continues with the literature on agents, agent properties and architectures, their methodologies and communication. As defined in Chapter 1, the Pre-assessment System is an agent based system. In view of that, this chapter looks at the various phases of agent oriented analysis and design for a choice of a suitable methodology for the design of the agent based pre-assessment system of this research. Also, the chapter discusses the *speech acts theory* (Searle, 1969) and its influence on agent communication languages, some agent oriented programming languages, and *Jason AgentSpeak Language* (Bordini, Hubner & Wooldridge, 2007) in the communication of logic based representation.

### 3.1 Agents

The term *agent*, otherwise known as agent based computing, agent based system or multiagent system, are increasingly used within information technology to describe a broad range of computational entities (Jenning & Wooldridge, 1995). An agent is an autonomous computer system that is situated in some environment (Wooldridge, 2009). In that environment agents exhibits properties of autonomy, sociability, reactivity and deliberation in order to meet their design objectives. Agents can observe and perceive the state of their environment, and can perform actions intended to change it (Fig. 3.1) (Russel & Norvig, 2003). The Figure 3.1 depicts the structure of an agent

model. In the model, agents have knowledge about the state of their environment, with sensors, agents can observe precepts or inputs, and select *condition-action* rules to act in that environment.

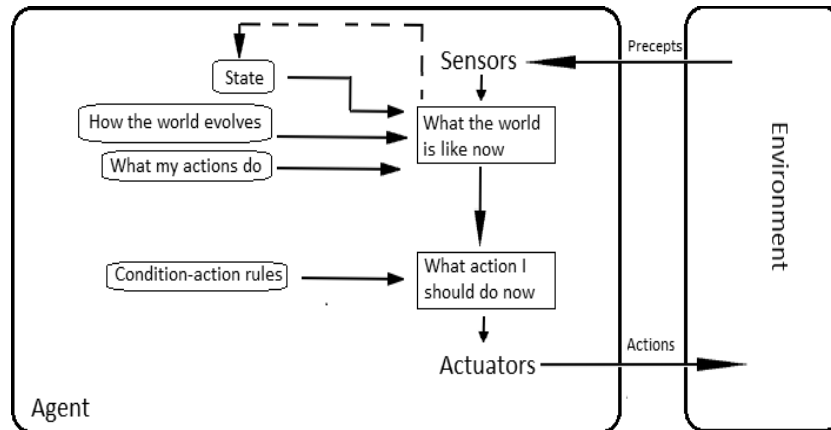


Fig.3. 1: The Structure of a Simple Reflex Agent (Russell & Norvig, 2010).

In Peredo *et al* (2011) agents are tools that independently perform various tasks on behalf of human user(s) or other software agents. Agent based system may not be stand-alone entities but a system consisting of a group of agents in the same environment otherwise known as a multi-agent system (Gladun *et al*, 2009). As applicable in other fields such as supply chain, autonomous vehicles, online trading, and healthcare delivery, multiagent systems are gaining wider recognition for educational applications.

Monett (2014), elaborated examples of agents' environment with features that are associated with teaching and learning. In Monett's illustration of the interactive tutor (Fig. 3.2), the *environment* that the agent will observe is specified as a set of students, the keyboard as *sensors*; and academic exercises, suggestion for materials and corrections as *actuators* on a display screen.

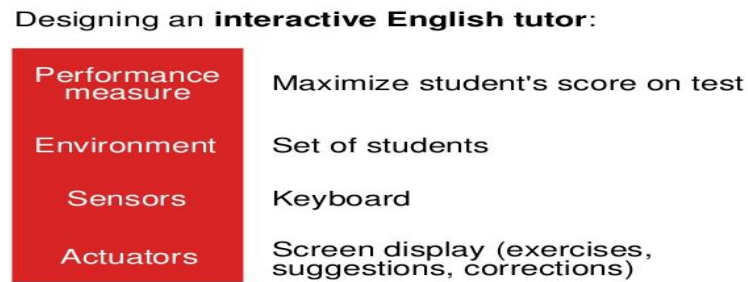


Fig.3. 2: Designing Intelligent Agents: An example (Monett, 2014).

### 3.2 Properties of Agent

Since agents independently perform different tasks on behalf of humans (Peredo *et al*, 2011), they also possess and exhibit some human attributes as described in literature. For example, Genesereth & Ketchpel (1994), Castelfranchi (1995), Goodwin (1995) Woodridge & Jennings (1995), Woodridge (2009), Padgham & Winikoff (2004), and Bordini, Hubner & Woodridge (2007) have all proposed that agents are:

- **Situated:** That agents exist in a world in which it has sufficient knowledge about, and can perceive and make changes to the world.
- **Reactive:** This is when an agent can perceive and respond to actions and changes in its world. This property become successful if the agent can respond quickly enough to the event. Failure to react leads to failure of subsequent goals. Reactivity of agents can be dual: response to percepts on a graphical user interface and/or response to shared messages.
- **Deliberative:** This is the application of practical reasoning mechanism on how to achieve a state of the world. A deliberative agent has an internal model of the world and uses its model to reason about the effects of perceived inputs in order to select appropriate intentions that it predicts will accomplish the task.

### 3.3 Agent Architectures

An architecture proposes a methodology for building an autonomous agent [system]; and explains how the system can be decomposed into the construction of a set of component modules [i.e. behaviours] and how these behaviours should be made to interact (Maes, 1991). In Wooldridge & Jennings (1995) agent architecture represents

the move from specification to implementation. The decomposition process in the views of Wooldridge & Jennings (1995) involves analysing the agent property to be satisfied, perception of input data, internal knowledge representation, and the programming language for implementation.

While Wooldridge & Jennings (1995) identified the different agent architectures, Chin *et al.* (2014) categorised the architectures into three broad groups, namely: cognitive architecture, semantic agent architecture and classical architecture. The classical agent architecture that comprise the logic-based architecture, reactive architecture, hybrid architecture, and BDI architecture are explained as follows:

### **3.3.1 Logic-based Architecture**

This architecture uses symbolic representation for modelling agent behaviour and reasoning. This involves the definition of agent capability using logic based semantics for expression of: rules, reasoning, knowledge preferences to react to several alternative choices of actions, and retrieval of information for a user's best interest (Dell'Acqua *et al.* 1999). De Silva (2009) asserted that logical formulas are used to represent agent beliefs, and from the deductions made from the logical formulas, agent behaviours are derived. That the deductions from the formulas are through a set of rules whose predicates or antecedents correspond to executable actions.

### **3.3.2 Reactive Architecture**

This is a direct *stimulus-response* approach. That is, percept-to-action that may change the state of the environment, and the dynamic beliefs of the actors or agents. Stimulus-response are agent behaviours i.e. plans which are used for decision making processes and for effecting changes in the agent environment for selective actions.

### **3.3.3 Hybrid Architecture**

This architecture is also known as *layered* architecture. It is a hybrid of the reactive and deliberative architectures. The subcomponents of the layered architecture are decomposed into hierarchies of layers to handle different behaviours that interacts.

There are two different modes of the layered architecture, namely; 1) horizontal layer, where all layers are directly connected to the input sensor and action output in the environment, and every layer functions concurrently (Fig. 3.3); and 2) vertical architecture, where the layers are arranged in sequence such that the data from the input sensor is transmitted from layer-to-layer until the final layer for action output (Fig. 3.4 and Fig. 3.5).

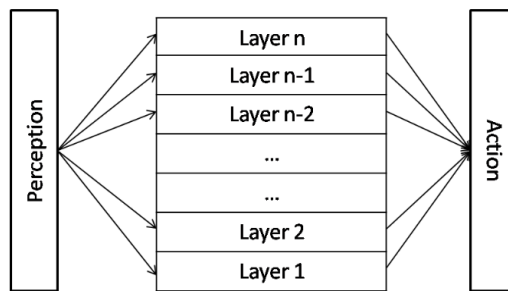


Fig.3. 3: Horizontal Architecture

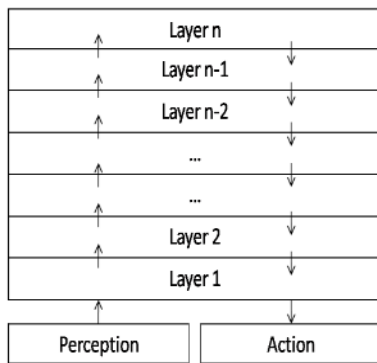


Fig.3. 4: Vertical architecture: two pass

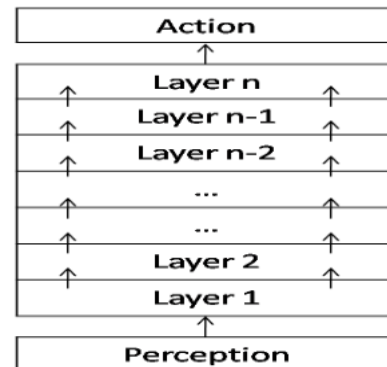


Fig.3. 5: Vertical architecture: one pass

### 3.3.4 BDI Architecture

This is a deliberative agent architecture based on mental states characteristic of agents which have belief, desire, and intention. *Beliefs* are the set of information an agent has about the world e.g. itself and the environment. *Desires* are the agent's motivation or possible options to carry out actions. Desires corresponds to *goals*, and are *post-conditions* executed in plans (Bordini, Hübner & Wooldridge, 2007). *Intentions* are the agent's commitments towards its desires and beliefs. Intentions are the executable

statements contained in an agent plan, and an unexecuted statement is a failed intention.

### **3.4 Agent Oriented Methodologies**

A Software methodology is a set of guidelines covering the entire life-cycle of a software development process. The set of guidelines that make up the software development stages have shared abstraction in both the Object Oriented Programming (OOP) methodology and Agent Oriented Software Engineering (AOSE) paradigm. The OOP developmental stages are Requirements, Analysis, Design, Development, Testing and Maintenance. While the AOSE process subsumes the steps in OOP methodologies, the concepts for developing objects (in OOP) are different from those in agent based systems. The OOP covers concepts such as objects, classes and inheritance. AOSE design concepts are terms that view agents as autonomous, situated, reactive, and social.

Several AOSE methodologies have been proposed and tested for application purposes. Amongst them are Gaia (Wooldridge *et al.* 2000), Tropos (Bresciani *et al.* 2004), MaSE (DeLoach *et al.* 2001), PASSI (Cossentino, 2005; Cossentino, & Potts, 2002), and Prometheus (Padgham & Winikoff, 2004). Though these methodologies show similarities, there are varying degree of differences in their respective design process: From requirements analysis through functionality modelling for agents to implementation. In the following section, the Gaia, Tropos and Prometheus are discussed.

#### **3.4.1 Gaia**

Gaia is a methodology that is based on the OOP analysis and design principles for modelling agent based system from the framework of a social organisation. From its organisational perspective, analysts can develop complex systems using a model that includes interacting entities and roles to achieve some set of organisational goals. A tool that supports the Gaia methodology is Gaia4E (Cernuzzi & Zambonelli, 2009).

The Gaia model is made of two major phases which are *analysis* and *design*. But with its concepts divided into two main categories: *Abstract* and *Concrete* concepts (Jennings, Wooldridge, & Kinny, 1998; Wooldridge, Jennings & Kinny, 2000). While the *Abstract* concepts are those used during the analysis stage to conceptualise the system, they do not have direct realisation within the system; the *concrete* components are those used in the design process, and do have direct counterpart during implementation.

Firstly, to begin the Gaia model, *Statement of Requirements* must be obtained before the analysis and design phase (Fig. 3.6). The statement of requirement is the identification of the domain problem of the system.

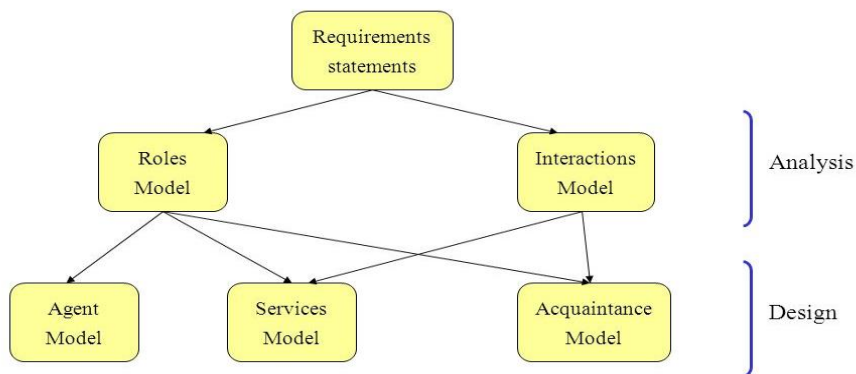


Fig.3. 6: The Gaia model (Wooldridge, Jennings & Kinny, 2000)

▪ **Analysis**

This is the phase where the structure of the systemic organisation needs to be understood given the requirement needs. Without details, roles (like offices) in an organisation, interaction between roles, and organisational goals are identified. The roles are defined by responsibilities, permissions, activities, and protocols (Wooldridge, Jennings & Kinny, 2000). In the analysis phase, the aim is to identify what (number of) agents will be part of the organisation given the decomposition of roles. Roles may be combined, and an agent can have multiple roles.

- **Design**

This is the stage where the roles, responsibility and interaction protocols that have been identified in the *analysis* phase are outlined between agents. *What agent does what, what agent interacts, and how?* At this stage *abstraction* starts to turn into *concrete* analysis that can transform into implementation. The design phase is made up of three models, namely: Agent Model, Services Model, Acquaintance Model (Wooldridge, Jennings & Kinny (2000):

1. **Agent Model:** The model that identifies and specifies the *agents* or *agent types* in the system. An agent type is a set of agent roles.
2. **Agent Services:** The model that identifies the main services of an agent role. A service is a coherent block of activity in which an agent will engage. Each service contains input, output, pre- and post-conditions.
3. **Acquaintances Model:** This is the description of the communication protocol (or links) between agent types. In this model, nodes represent agents while links which are directed graphs represent communication between nodes. For example,  $a \rightarrow b$  which means *agent a* is *sending* message to *agent b*.

### 3.4.2 Tropos

Tropos is an agent oriented programming (AOP) methodology that strongly emphasise two key notions: The use of mentalistic features such as *goals and plans* from the BDI model, and *Early requirement analysis* (Bresciani *et al.* 2004). The tool, Taom4E (Morandini *et al.* 2011) is a graphical modelling editor that supports the Tropos methodology development phases. In Tropos, there are five main development phases (Bresciani *et al.* 2004):

- **Early Requirement**

This is the first phase of requirement analysis held to be crucial compared to the *Later prescriptive requirement* phase. In this phase, the ideas developed are used in the later requirement phase. The domain stakeholders (or entities) are identified, conceptual



models are developed, and social *actors* are modelled so as to achieve organisational goals, furnish resource, and execute plans.

- **Later Requirement**

The analysis from the Early phase are engaged at this phase. Conceptual models are also extended. The aim of the requirement phases is to provide functional requirements for the system.

- **Architectural Design**

At this stage, the system underlying architecture is defined in terms of subsystems (i.e. components or actors), and inter-connected through control flow. The system actors are mapped to set of agents, each with their specified functions.

- **Detailed Design**

This phase specifies agent capabilities and interactions between agents. At this stage the implementation platform can be chosen where detailed design can be mapped directly to the code.

- **Implementation**

This is the step-by-step activity carried out for the realisation of the system on the programming or development platform.

### 3.4.3 Prometheus

Prometheus (Padgham & Winikoff, 2004) is an AOSE methodology designed for the realisation of BDI agent systems with the use of goals and plans. It supports development activities from requirements specification through to detailed design for implementation. Prometheus has three inter-connected design phases which are *System Specification*, *Architectural Design*, and the *Detailed Design* (Fig. 3.7). Prometheus Design Tool (PDT) (Padgham *et al.* 2008; Zhang *et al.* 2008) is a graphical editor that supports the Prometheus methodology.

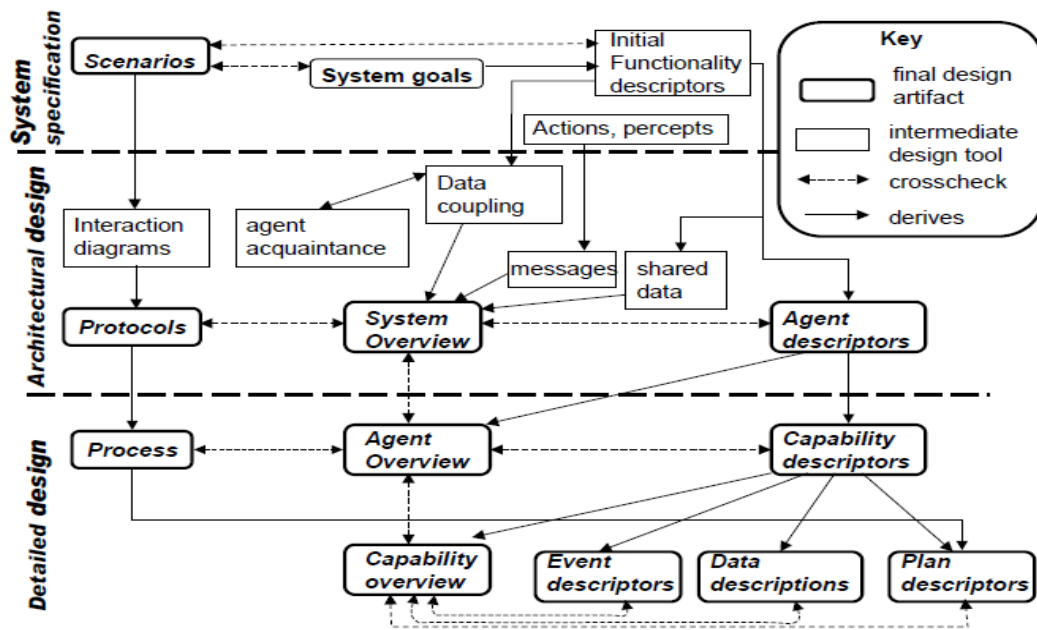


Fig.3. 7: The phases of the Prometheus methodology (Padgham & Winikoff, 2004)

The PDT is an AUML (Agent Unified Modelling Language) tool and graphical editor that supports the development and documentation of the major phases of the Prometheus methodology for building agent based systems.

- **System Specification**

This is a major phase that characterises the definition of the scenarios, goals, roles and the expected interactions within the system. This phase also identifies the interface of the system, incoming percepts, and actions or outgoing information. In the PDT tool, some of the facilities for realising the specification phase are Scenario Diagram, System Goal Diagram, and System Role Diagram.

- **Architectural Design**

This is the phase where the agent types, their roles, the data and the kind of communication and messages that the agents will involve in are identified. At this phase, the system overall structure is already constructed and scenarios are developed into goals, then to roles and interactive protocols. When developing goals, Zhang, Kendall, & Jiang (2002) states that the question to ask is: *what is to be done* and *how*

they can be done? The PDT tool supports the architectural design phase with the System Overview Diagram.

▪ **Detailed Design**

This phase defines the design of individual agent and their internal structure in terms of *Capabilities* descriptors which are a set of related plans used for achieving a common goal or common set of goals. Other descriptors are for data, events and plans. At this phase, much finer details from the architectural phase are established. The PDT tool supports the detail design phase with facilities such as Agent Overview Diagram (Fig 3.8).

	<i>Dynamic Models</i>	<i>Structural Overview Models</i>	<i>Entity Descriptors</i>
<i>System Specification</i>	Scenarios	Goals	Functionalities actions & percepts
<i>Architectural Design</i>	(interaction diagrams) Interaction Protocols	(coupling diagram) (agent acquaintance) System Overview	Agents Messages
<i>Detailed Design</i>	Process Diagrams	Agent Overview Capability Overview	Capabilities Plans, Data, Events

Fig.3. 8: Major models of Prometheus (Padgham and Winikoff, 2002)

PDT support for implementation, testing, and debugging is still limited (Padgham & Winikoff, 2004). Thus, interaction design accomplished with the PDT tool have had their implementation carried out on different agent oriented programming (AOP) platforms. For instance, the Electronic\_Bookstore system (Padgham & Winikoff, 2004) was implemented on JACK<sup>TM</sup> (AOS, 2015), Bordini, Hubner & Wooldridge (2007) version of the Electronic\_Book was implemented using Jason, and the Gold Miners robot (Bordini, Hubner & Tralamazza, 2006) implementation using Jason. The PDT also supports Jack<sup>TM</sup> skeletal code generation in Java (Fig. 3.9).

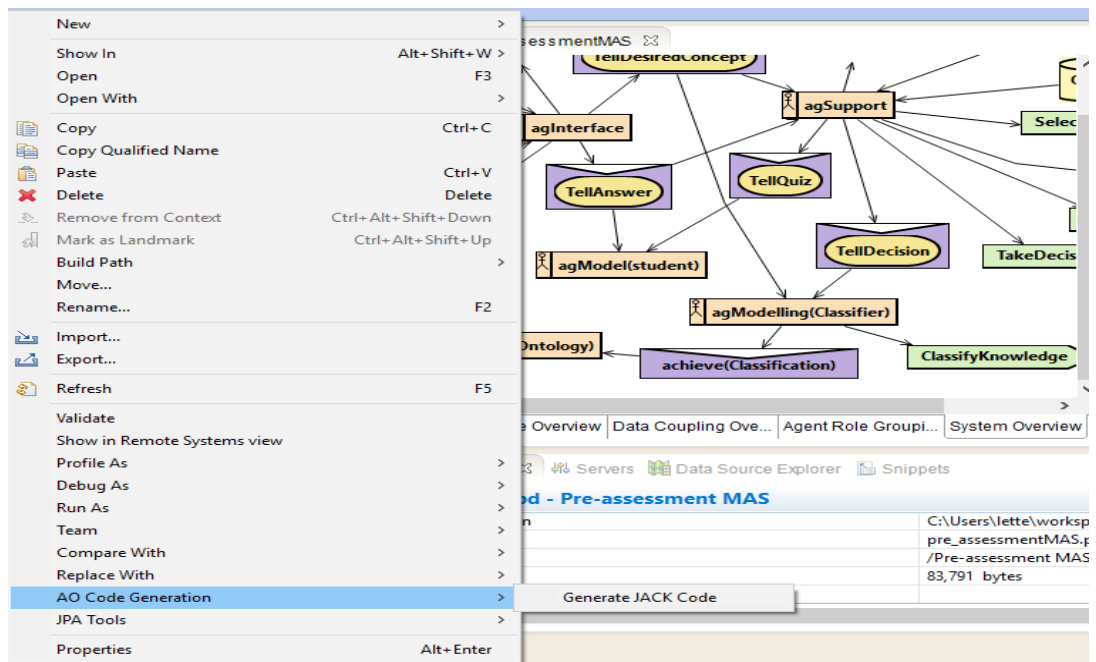


Fig.3. 9: Jack code generation screen shot. The code generated are in Java, which is not the language chosen for the execution of one of the objectives of this research.

### 3.5 Comparison of AOSE Methodology

The Figure 3.10 is the highlights of the Gaia, Tropos and Prometheus AOSE methodologies. The Figure depicts the similarities and differences in their design phases. The similarities centres around the use of a customised design tool for MAS development, but all differ in the design steps. The *Tropos* concept of *Softgoals* which is equivalent to *Subgoals* in Prometheus is a breakdown of *Hardgoals* and *Initial goal* of agents (or actors) functionalities, respectively.

Methodologies	Phases	Comparison
Gaia	<ul style="list-style-type: none"> <li>* <i>statement of requirement</i></li> <li>* <i>analysis</i></li> <li>* <i>design</i></li> </ul>	<ul style="list-style-type: none"> <li>* Lack detailed step-by-step breakdown.</li> <li>* No details on how requirement statements may be acquired.</li> <li>* View agent system as an organisational model.</li> <li>* Roles are similar to functionalities in Prometheus.</li> <li>* Editor tool Gaia4E supports design.</li> </ul>
Tropos	<ul style="list-style-type: none"> <li>* <i>early requirement phase</i></li> <li>* <i>later requirement phase</i></li> <li>* <i>architectural design</i></li> </ul>	<ul style="list-style-type: none"> <li>* Emphasises the <i>Early Requirement Analysis</i>, then the <i>Later Requirement Phase</i>.</li> </ul>

	<ul style="list-style-type: none"> <li>*detailed design</li> <li>* implementation</li> </ul>	<ul style="list-style-type: none"> <li>* Specialisation of Goals into subclasses of <i>Hardgoal</i>, and <i>Softgoals</i> for actors of system.</li> <li>* No general architecture containing all the phases of design as in Gaia, MaSE, or Prometheus.</li> <li>* Has a design support tool called Taom4E.</li> </ul>
Prometheus	<ul style="list-style-type: none"> <li>* system specification</li> <li>* architectural design</li> <li>* detailed design phase</li> </ul>	<ul style="list-style-type: none"> <li>* No Early Requirement phase as in Tropos. But this can be adapted.</li> <li>* Uses <i>Initial goals</i>, that are refined or broken down into <i>Subgoals</i> for agents.</li> <li>* Very detailed design activity from System Specification phase to other phases.</li> <li>* Reliance on expert knowledge on domain subject for requirement acquisition.</li> <li>* Has a customised PDT, a AUML tool that supports design process.</li> </ul>
<b>Methodologies</b>	<b>Phases</b>	<b>Comparison</b>

Fig.3. 10: Comparative summary of Gaia, Tropos & Prometheus.

### 3.6 The Speech Acts Theory

When we use utterances in a language our intention is often to achieve a specific goal that is reached by a set of actions (Finlay & Dix, 1996). The acts that we perform with language are called *speech acts* (Austin 1962; Searle 1969). Speech acts theory treats communication as actions. This is on the premise that speech actions are performed by agents just like other action in realising their intentions (Woodridge, 2009).

#### 3.6.1 John Austin: 1962

In the use of words which make up sentences, there is a meaning (i.e. semantics) as a result of the relationship between the words (i.e. structure or syntax). Every utterance has the characteristics of *actions* (things we do) (Woodridge, 2009). A speaker performs a speech act by uttering a sentence with an associated intention to the hearer (Oishi, 2006). The actions performed could change our state of belief, the physical world or environment.

This concept of speech acts is recognised to have begun with John Austin in 1962. Austin (1962) investigated three different aspects of speech acts that can form

*performative* verbs, namely: *locutionary*, *illocutionary*, and *perlocutionary* acts which are known as the stages of sentence transition. A sentence starts with *locution* (an utterance), goes through *illocution* (the performative action) and end with *perlocution* (the effect of the action). The illustrations are given as:

- **Act (A)** or Locution (*Utterance*): He said to me ‘make some cake’. The act of saying something i.e. the utterance is heard.
- **Act (B.a)** or Illocution (*Request*): He ‘urged me to make me some cake’. The act performed in saying something, i.e. belief addition.
- **Act (B.b)** or Illocution (*Command*): He ‘ordered me to get some cake’. Also the act performed in saying something i.e. also belief addition.
- **Act (C)** or Perlocution (*Effect*): ‘He got me to make cake’. The act performed after the Saying.

In agent technology and programming in general, *locution* (e.g. giving information) is the act of variable initialisation, declaration or a *tell* performative; and *illocution*, the request by message passing or input statements such as *get*, *askOne*, *achieve*; while *perlocution* is the output after processing. The *performative* begins from the issuing of utterances to the performing of the action. Thus in utterances, the performative verb is action or *doing* words succinctly denoted and are capable of instigating a course of action or changing the state of things. Examples are *broadcast*, *tell*, *askOne*, and *achieve* in agent communication technology.

For successive completion of performatives, three “*felicity condition*” conditions are required (Austin, 1962:14; Woodriddle, 2002:165):

1. There must be an accepted *conventional procedure* for the performative, and the circumstances and the actors (or agents) must be as specified in the procedure.
2. The procedure must be *executed correctly* and completely.
3. The act must be *sincere*, and any *uptake* required must be completed, insofar as is possible.

Austin (1962) then classifies *illocutionary* acts into five types, namely:

- i) **Verdictive:** *one can exercise judgment;*
- ii) **Exercitive:** *exert influence or exercise power;*
- iii) **Commissive:** *assume obligation or declare intention;*
- iv) **Behabitive:** *adopt attitude, or express feeling; and*
- v) **Expositive:** *clarify reasons, argument, or communication.*

Although it is often argued that Austin's classification is not complete and that those coined categories are not mutually exclusive (Oishi, 2006). In other words, they are overlapping categories (Jiang & Huhns, 2005).

### 3.6.2 John Searle: 1969

John Searle, who inherited his idea from John Austin, elaborated on the Speech Acts Theory; and proposed five but varied classification of *illocutionary* speech acts to Austin's (1962), namely:

- i) **Assertives:** *Telling people how things are;*
- ii) **Directives:** *getting them to do things;*
- iii) **Commissives:** *committing ourselves to do things;*
- iv) **Expressives:** *expressing our feelings and attitude; and*
- v) **Declaratives:** *bringing changes into the world by our utterances.*

Searle (1969) points out that, *to perform an illocutionary act is to express an illocutionary intention* (Searle 1969) using performative verbs such as *state, request, command, order, and promise* (Searle, 1969:23). This is a variation from Austin's (1962) that in the performative: *the issuing of utterances is the performing of an action* (Austin, 1962:6). In actual fact, not all actions are performed after perceiving or hearing of the utterance. Humans and agents are alike, they have autonomy—*To or Not To*—over their behaviour.

From the foregoing, let a speaker *S* utters a sentence *T* to a hearer *H*, ACTION *A* can only be performed by *H* after the occurrence of *T* if and only if *H* understands the

sentence or message from  $S$ , and  $H$  has the capability to act (Searle, 1969, 57:61; Woodriddle, 2002:165).

If intelligent systems are to interact with humans or other agents, then speech acts performatives must be part of their program designs, and the acts treated as physical actions (Woodriddle, 2009). The sender's [e.g. a user] intention must produce certain *response*  $r$  in the receiver [e.g. situated agent in the artifact], and a value [e.g. concept] of  $r$  [when received] (Schiffer, 1972) that would change its mental state. With speech acts performatives, agents would share the knowledge contained in a message.

### 3.7 Pre, Post & Completion Conditions

The speech acts theory of John Austin and John Searle have predominantly influenced the development of Agent Communication Language (ACL) such that current speech-act based ACLs specify domain knowledge representation and performative communication acts. Labrou & Finin (1998) semantics of speech acts shed more light on the *locutionary*, *illocutionary* and *perlocutionary* acts. These three performative conditions for agents' communication have been represented as *preconditions*, *postconditions* and *completion* conditions (Labrou & Finin, 1998; Bench-Capon, 1998):

- **Preconditions:** The fact that is established before an act is performed (i.e. utterance).
- **Postconditions:** The fact that is established after the act is performed (i.e. action).
- **Completion:** The fulfilment of the intention of the act performed (i.e. effect).

### 3.8 Agent Communication Languages

Communication between entities comes by interaction of information when there is an utterance of a concept i.e. word, phrase, or sentence at one end and perception at another. In a MAS environment, communication is a rational behaviour between agents using a conventional language (Russell & Norvig, 2003). Thus, communication



is realised by a set of syntactic definition and semantic rules specified in a given programming language, used in a program.

According to Pitkäranta (2004) agent communication can be divided into two fundamental parts. Firstly, that agents have to agree on a common agent communication language, which defines the types of the message performatives and their meanings. Secondly, agents must have a common understanding of the knowledge that is exchanged within the messages. In that regard, Dogac & Cingil (2003) asserted that smooth MAS communication broadly depends on three composite layers (Fig. 3.11), namely:

- Agent Communication Language e.g. Knowledge Query and Manipulation Language (KQML) which uses performatives such as the *tell*, *achieve*, and *askOne*;
- Content Interchange Format i.e. the content language e.g. KIF, Prolog; and
- Ontology i.e. the knowledge domain of interest for the system.

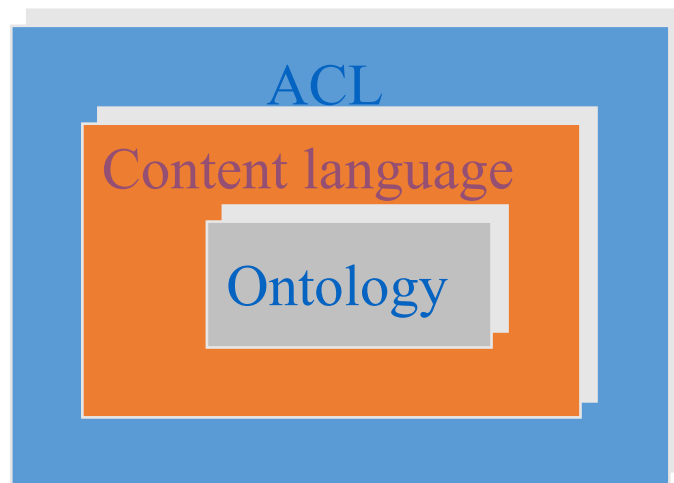


Fig.3. 11: Components of Agent Communication Language (Dogac & Cingil, 2003)

### 3.9 Agent Oriented Programming languages and Platforms

Agents are developed or programmed from a variety of different programming languages or platforms. The following section presents a range of agent oriented programming (AOP) and platforms for developing agent, their support capability for building and implementing agent based systems.

### 3.9.1 Agent0

Agent0 is a simple agent oriented programming (AOP) language for implementing a multiagent system (Shoham, 1991). In agent0, an agent is defined to have four parts: i) a set of capabilities (describing what the agent can do: a relation between an agent's mental state and environment), ii) a set of beliefs, iii) a set of commitments or intentions, and iv) a set of commitment rules containing a message condition, a mental condition and an action (Bădică *et al.* 2011). Agent0 agents communicate via *request* to performing an action, *unrequest* to stop an action, and *inform* that changes the agent's belief.

### 3.9.2 PLACA

PLACA is the improved version of Agent0. PLACA was the first language to introduced the concept of *plans* in agents. Both Agent0 and PLACA were designed for experimental use, not for practical applications.

### 3.9.3 GOAL

GOAL is an agent programming language that uses declarative knowledge to specify what the agents wants to achieve. GOAL provides building blocks to design and implement *rational* agents. An agent *beliefs* and *goals are used for action selection and structured decision making*. Agents use knowledge representation language (symbolic, logic language) to represent information they have, their belief, or knowledge in the environment in order to achieve their goals. Programming an agent in GOAL means to program with the *mental state* of the agent and providing a coding strategy for action selection. A mental state consists of declarative knowledge, beliefs and goals (GOAL, 2016). Applications developed on GOAL has been in transportation and logistics domain. Goal has no support for inter-agent communication via speech acts.

### 3.9.4 Soar

Soar (Laird, 2008; Laird, 2015) is an architecture for developing general intelligent systems. Soar represents and uses declarative knowledge (i.e. known facts). In the area

of teaching and learning, Soar has been used as a platform for the development of STEVE (Soar Training Expert for Virtual Environments) an animated pedagogical agent (Johnson & Rickel, 1997). STEVE teaches students procedural tasks, for example, how to operate controls in an engine room. The capabilities of STEVE include observing the state of the world, monitors students' requests and questions posed by students. The STEVE system has specified knowledge which it uses to execute actions in the form of a hierarchy of plans. Each plan includes a set of steps, a set of ordering constraints, a set of casual links of steps that leads to the achievement of goals that is either an end goal or a set of pre-condition for another subtask. The Soar architecture does not support the BDI model and speech-acts based communication in agent based applications.

### 3.9.5 JACK

JACK™ is a commercial agent framework for developing autonomous decision making system by the Agent Oriented Software (AOS). JACK is a BDI based language that is based on Java (Busetta *et al.* 1999). JACK supports the development of multiagent and agents exchange messages interchangeably in a peer-to-peer mode. JACK agents are not bound to any specific agent communications language (Howden *et al.* 2001). In Jack, plans constitute reasoning methods that provides agent the capability to act. Examples of applications developed on JACK are in decision support, and defence operations. As a commercial agent development platform, Jack is a costly software; and it is suitable alternative to implementing the pre-assessment system.

### 3.9.6 Jadex

Jadex is a Java- based agent middleware architecture that implements the BDI agent model: *beleifs*, *desires* (goals in JADEX) and *intensions* (plans in Jadex) (Bădică *et al.* 2011). Jadex does not enforce a logic-based representation of belief (Braubach *et al.* 2004). Jadex uses object-oriented programming for belief representation, and declarative and procedural approach for specifying and defining agent components. The Jadex agent are able to run on Jade. Like Jade that is also a middleware architecture, Jadex agents communicate by exchanging Agent Communication

Language (ACL) messages. This also make Jadex a suitable platform to implement the pre-assessment system where agents can have autonomous control over their state.

### **3.9.7 Jade**

Jade (Java Agent Development Framework) is a FIPA compliant software architecture for developing agent applications and interoperable intelligent mulitiagent systems (Bellifemine, Poggi, & Rimassa, 1999; Bellifemine, Caire & Greenwood, 2007). Jade is considered to be agent middle-ware that implements an Agent Platform for distributed systems across networks. Agent communication is through message passing in textual form, and FIPA standard is that the Agent Communication Language (ACL) which is close to KQML is the language for inter-agent interaction and interoperability on Jade. Running Jason agent language on the *infrastructure Jade* initialises the Jade Agent Management platform. Thus, Jade is a suitable platform in which the pre-assessment system agents can be implemented.

### **3.9.8 AgentSpeak**

AgentSpeak programming language is a natural extension of logic programming for programming BDI agents. An AgentSpeak agent is created by the specification of a set of beliefs which is a set of ground (first-order) atomic formulas and a set of plans which forms its plan library. The set of beliefs are the initial state of the agent's knowhow of its world. The belief atoms in first-order predicate form are belief literals (Bordini & Hubner, 2007; Bordini, Hubner & Tralamazza, 2006). For instance, *father(peter)* (Baadar & Nutt, 2003) and *member(sam, cs)* (Gelfond, 2008) are unary predicate and binary relations, respectively. An AgentSpeak plan has a head which consist of a triggering event that indicates the event in which a plan will be relevant, and conjunction of belief literals in predicate form representing a *context*, and a plan body which is a sequence of actions or goals that the agent has to achieve or test.

### **3.9.9 Jason Agent Language**

Jason is an extended version of the AgentSpeak language. In other words, a Java based interpreter of AgentSpeak. It is an agent-oriented logic programming language whose

syntax draws similarities with Prolog (**P**rogramming in **l**ogic) language (Bădică *et al.* 2011) for belief representation and query. Jason implements the operational semantic of AgentSpeak in the programming of MAS. Jason allows programming of agents in the BDI model, environment perception, belief updates, inter-agent messages or communication, and use of *knowledge on how to do things* in the form of *plans*. Agents are programmed using beliefs, intentions and sub-goals in plans to accomplish goals. Beliefs representation in Jason is in FOL atomic facts.

Programming in Jason is *procedural* (plan by plan selection), *declarative* (initial specification of beliefs and goals like in Prolog) (Bordini, Hubner & Wooldridge, 2007). In Jason, agents communicate with each other in high-level manner based on the *speech acts* (Searle, 1979) theory. Jason is also tightly integrated with Java such that Jason can be used to situate agents in an environment model that is developed with Java. Jason is cross-platform API that can be configured and run on jEdit or Eclipse IDE.

The type of *Infrastructure* determines the nature of environment in which a MAS will run or situate. As *Open Source* software, Jason allows developers to program multi-agent systems using the *Centralised*, or *Jade* Infrastructure.

- **Centralised:** This is the infrastructure that allows MAS to run within a localised system or computer. The *Centralised Infrastructure* which is specified as

Infrastructure: Centralised  
runs Jason MAS Project on a local machine.

Recall that one of the objectives of this research is *to investigate the communication of ontological concept (i.e. FOL atomic formulas) in the process of identifying gaps in students' learning*. Before logic based formulas are communicated or shared by agents for the identification of gaps in a learning domain, structured knowledge is represented in FOL in agent as beliefs. The beliefs in Jason agent programming language are in FOL form. That is, beliefs can be unary predicate or binary predicate relation such as  $p(a)$  or  $p(a, b)$ , respectively. Also Jason is a *speech act* (Searle, 1979)

based language that supports inter-agent communication in a MAS paradigm. In Jason KQML performatives such as *tell*, *askOne*, and *achieve* are used for communication between agents. While KQML is adequate for simple message passing, Cost *et al.* (1999) observed that it would however break down as the range of interaction that an agent will partake increases. Nonetheless, KQML performatives such as *tell* support semantic interoperability and knowledge sharing of concept and resource between agents (Klapiscak & Bordini, 2009; Da Silva Vieira, 2007). The TABLE 3.1 below presents a comparative analysis of the foregoing AOP languages and platforms, and our informed choice of Jason for implementing this project.

**TABLE 3. 1: COMPARISON OF AGENT ORIENTED PROGRAMMING (AOP) AND PLATFORMS**

AOP	BDI	Speech acts	Logic based	Declarative	Procedural	Java based	Agent interaction	Open source
Agent0	✓	✓		✓			✓	✓
PLACA	✓	✓		✓			✓	✓
GOAL	✓		✓	✓				✓
SOAR	✓	✓		✓	✓			✓
Jack	✓	✓				✓	✓	
Jadex	✓			✓	✓	✓	✓	✓
Jade	✓	✓				✓	✓	✓
AgentSpeak	✓	✓	✓	✓	✓		✓	✓
Jason	✓	✓	✓	✓	✓	✓	✓	✓

### 3.10 Agent Interaction in Jason

Communication in MAS is typically based on the *speech act* paradigm (Bordini, Hubner & Wooldridge, 2007). For inter-agent communication, there must be a sender, a receiver, the performative and the content as shown in the construct:

*<sender, illoc\_force, propositional\_content>*

where the *sender* is an AgentSpeak atom (i.e. a simple term), meaning the name of the agent that sends the message; *illoc\_force* is the performative, the intention of the

sender; and *propositional\_content*, the act to accomplish (Bordini, Hubner & Wooldridge, 2007). The above construct are only executable as part of a plan. Thus the message structure of the sender agent is given in the format:

*.send<receiver, illoc\_force, propositional\_content>*

Before looking at the meaning of a *plan*, some agent oriented programming (AOP) concepts as they pertain to Jason are first discussed.

### 3.10.1 Beliefs

Beliefs in Jason are logic based representation that holds the knowledge an agent has about the world. One agent can perceive the world and another can update the world. Every agent has a *beliefbase* (BB) that contains the *beliefs* or mental status of the agent at a given point in time. In other words, BB are a knowledge base (KB). A KB is a set of sentences (Russel & Norvig, 2010) or information—semantic literals that agents can understand and communicate. Thus, beliefs are assertion of the agent’s knowledge about its world or environment. They are represented in predicate logic in the form:

*predicate(object)*

or

*predicate(subject, object).*

Some of examples of beliefs representation are (Bordini, Hubner & Wooldridge, 2007):

`blue(box1) .`

Stating that *box1* has the colour *blue*, and

`fact(0, 1) .`

Which states that the factorial of 0 is 1. These are beliefs an agent programmer would provide as initial beliefs.

### 3.10.2 Annotations

These are terms that provide detailed information that are strongly associated with a particular belief, and they are enclosed in square brackets. Generally, they can be represented with extended annotation given in the form:

*functor(term<sub>1</sub>, ..., term<sub>n</sub>)[annotation<sub>1</sub>, ..., annotation<sub>m</sub>].*

Where  $annotation_i$  are first order terms. For example, (Bordini, Hubner & Wooldridge, 2007):

```
red(box1 [source (percept)]).
```

This type of annotation depicts to the agent that the information is perceived from the environment.

Or

```
blue(box1) [source (ag1)].
```

which states that the belief source is the agent  $ag1$ .

Other kind of beliefs annotation is that which is appended to a set of related beliefs that are initialised as a group of related terms that belongs to one knowledge domain.

This Klapiscak & Bordini (2008) called semantically enriched (SE) literal e.g.

```
hasRating(hilton, threeStarRating) [o(travel)].  
isPartOf(wembly, london) [o(travel)].
```

that asserts that `hilton` which is an individual in the relation is related to `threeStarRating` by the object property `hasRating`, and that the individual `wembly` is related to the `london` individual by the `isPartOf` object property, respectively; where the annotation specifies that both relations are of the travel `[o(travel)]` ontology.

### 3.10.3 Goals

Goals can be considered as events that needs to be achieved. They are the part of a plan that makes the entire plan to be fulfilled or completed. In other words, *goals* are the *post-condition* of a plan (Bordini, Hubner & Wooldridge, 2007). Generally, in Jason, there are two types of goals:

- *Achievement Goals: Achievement goals* are those prefixed by the ‘!’ operator and they are *goals to do*. The syntax is

```
!achievement goal.
```

Example:

```
!write(book).
```

Which is assigning the *goal* to write a book.



- *Test Goals: Test goals* are those prefixed by the ‘?’ operator and are *goals to test* the truthness of a belief in order to retrieve the information from BB. The syntax form is

*?test goal.*

Example:

?publisher (P).

That *tests* whether *P* is a publisher.

### 3.10.4 Mental Notes

At runtime or MAS execution, agents are also able to create beliefs and add them to their BB. These kinds of dynamically-created beliefs are referred to as *mental notes* which may be updates as a result of the changes that has occurred in the environment they are part of, arithmetic operations performed, or messages (also known as percepts) passed by other agents. The operators *-+* are used to make mental notes. An example is

*-+current\_targets (NumTargets) ;*

which updates the current number of targets *NumTargets*. The meaning of this logic formula can be split into two: *-current\_targets (NumTargets) ;* which is to delete information about any previously stored beliefs (if there exists one) about number of targets, and *+current\_targets (NumTargets) ;* which is to add a new number of targets to beliefs.

### 3.10.5 Internal Actions

These are actions that are executed from within the *body* part of an agent, not from the environment. In this process, the whole action will be done as one step of the agent’s reasoning cycle. *Standard internal action* has the full-stop, that is ‘.’ prefix to a statement. A few standard internal actions are:

- .send used for inter-agent knowledge communication.
- .print for screen display of information.
- .wait which suspends an intention for a specific time.
- .date that gets the current date.

.concat which is used for concatenating (i.e. joining strings).

### 3.10.6 Plan

Each agent is an autonomous entity with several plans (list of courses of action). In executing a plan, agents make a selective choice, each in turns. Upon the receipt of a percept or message, a selection is made from amongst these plans for the appropriate action to execute. A plan has three distinct parts: *triggering\_event*, *context*, and *body*, and structure as:

$$triggering\_event : context \leftarrow body.$$

- The *triggering\_event* defines the occurrence of an events that can initiate the execution of a plan.
- The *context* is the pre-condition that states what the agent already knows, which are beliefs in first order or predicate terms that must be true for a plan *body* to be executed. It is the context that decides what plan is likely to succeed. In technology enhanced learning (TEL) for recommendation systems, context is also defined as any information that can be used to characterise the situation of an entity such that the term entity refers to a person, place or object (Dey, Abowd & Salber, 2001; Verbert *et al.* 2012).
- The *body* are series of atomic operations or set of actions that the agent can perform. In the performance of these actions, beliefs are updated, environment status are changed, and other agents are communicated. *Internal actions* as listed above are carried out in the body of a plan. A plan *body* also have *goals* and *sub-goals* that executes the intention of the plan.

An example is (Bordini, Hubner & Wooldridge, 2007):

```
@h3
+!has(owner, beer) : too_much(beer) & limit(beer, L)
  <- .concat("The Department of Health does not allow
  me ", "to give you more than ", L,
  " beers a day! I am very sorry about that!" ,M);
  .send(owner, tell, msg(M)).
```

where,

@h3 is the plan label that is giving a name to the plan. The +!has(owner, beer) is the *triggering\_event* adoption from a previously stated *achievement goal* !has(owner, beer). The too\_much(beer) & limit(beer, L) are the pre-conditions in the plan *context* that needs to be true. A plan *context* can also contain negated facts to test as a pre-condition. Or a comparison operator == (for equal) or \== (for different) that is comparing two terms like in Prolog. The .concat() predicate or functor is the agent action in the plan *body*, which is *concatenating* the sentences in quotes, and to store in the variable M. The .send() is another agent action that is communicating with the agent owner using a tell performative to inform the agent of the content of M.

### 3.10.7 Why Jason Agent Language?

Agents are computational entities that can be situated in simulated environment or in a real world. In this work, multiagents are meant to interact and to perceive the real world. For instance, consider a MAS developed to control the temperature of a room under the condition of observable number of people at any given time. When an agent acts, the action will be effected by a heating device (i.e. the hardware) and its percepts by a sensor also in the heating device. Such environment functionality can be supported by Java in developing the software side of the agent interface that enables the agent to continuously observe the environment.

To program a MAS for educational purposes, the choice of Jason was informed based on the analysis of the preceding subsections and the Table 3.1 above. More so, in Jason, agents can be programmed to have individual responsibility and cooperate on tasks through inter-agent communication. As a reactive system, Jason agent language applies practical reasoning approach to agent actions such that agents can continuously monitor their environment, update their beliefs and take action according to the *context* of their plans. Agents' observation of their environment can be synchronous or asynchronous. In this study and system research, agents' observation of their environment shall be asynchronous via the CartAgO artifact (Ricci, Piunti, Viroli, 2011).

### 3.11 Agent Environment Programming

One of the properties of agents as given earlier is that they reside in an environment from where they get percept through sensors, and there-after act on them via actuators (Wooldridge, 2009; Russell & Norvig, 2010). In a MAS, such an environment or percepts from it are shared by agents (Bordini, Hübner, & Wooldridge, 2007). An environment can be a real world (e.g. in manufacturing) or a simulated world (i.e. virtual). Environments can either be fully observable or partially observable by the agents. For instance, a world where an agent is directly situated and can observe the dynamic changes in it is a fully observable environment e.g. the domestic cleaning robot (Bordini, Hübner, & Wooldridge, 2007). But where agents cannot be directly situated in an environment to observe it, yet can perceive inputs from such environment is what Wang (2014) referred to as Partially Observable state. In Wang (2014) development of an ITS students were termed as the partially observable environment for agent observation. The environment in this research is as conceived in Wang (2014), where the *partially observable* environment is not the natural environment such as in the domestic cleaning robots, but an environment in the context of AOSE where the environment is part of the software system: This, Ricci, Piunti & Viroli (2011) called *endogenous*. From this viewpoint, Ricci, Piunti & Viroli (2011) states that

$$\textit{Programming MAS} = \textit{programming agents} + \textit{programming environments}$$

with the view that the two sides of the equation are programs, but with the environment programming part strongly integrated to the agent part. This critically conforms to the definition of an agent in Wooldridge (2009) that — an agent is a computer system that is situated in some environment.

#### 3.11.1 Artifacts and Human Interaction

The term *artifact* was first introduced by Ricci, Piunti, & Viroli (2011) as an interface for human-agent interaction design, and state that artifacts are runtime devices providing some kind of function or service which agents can fruitfully use both individually as an agent and collectively as multiagents to achieve their individual as well as social objectives. Artifacts can be generally conceived as function-oriented

computational devices in which function refers to the meaning that is generally used in human sciences such as sociology and anthropology, as well as artificial intelligence (AI) to depict the purpose for which the device has been designed. Which is to support agent activities in observing percepts or inputs and display of outputs. Artifacts from a MAS programmer point of view are a first-class abstraction that will target and program a functional environment that agents can exploit at runtime. This includes functionalities that concern observation, inter-agent interaction, and interaction with the external environment. Artifacts are tools that supports agents and humans to achieve their given goals and needs, respectively. This is achieved by the construction and configuration of a common interface between agents and human users. Artifacts are agent's sensors for obtaining input states that can trigger the action of the agent or MAS.

### 3.11.2 The CArtAgO Artifact

The CArtAgO framework (Common Artifact infrastructure for Agent Open environment) (Ricci, Piunti, Viroli, 2011) is a model for realising environment-mediated interaction between agent and/or human. The *MySimpleGUI* interface (Ricci, Piunti, & Viroli, 2011) is one example of an agent based graphical user interface (GUI) implementation from the CartAgO framework. At the start of the MAS, the agent creates the GUI which is the interface for the user and *agent* system to interact. During operation, which are iterated numeric calculation, the agent-designate on the artifact monitors events that are programmed in Java as input (from mouse click actions) and output the processed results.

### 3.12 Summary of Chapter

As a continuation of the literature survey, this chapter presented the structure of the simple reflex agent model, and an interactive tutor agent model. It presented and described agents as computer system that react to events in their environment, and cooperative through interaction to solving a problem, deliberative before the selection of a plan for execution, and autonomous because they have control over their internal actions. The chapter presented three categories of agent architectures and stated that

the classical architecture comprises the logic-based, reactive, hybrid (which combines both the reactive and deliberative models); and BDI architecture modelled after the human cognitive status. The chapter went further and surveyed agent methodologies: Gaia, Tropos and Prometheus in their phase to phase descriptive designs. Though all three mentioned methodologies have their associated design tools, Prometheus Design Tool (PDT) appears to be more detailed for developing agent based systems. The *speech acts* as a theory of semantic (meaning) communication was stated to have influenced agent communication or interaction languages. Different types of agent programming languages were also covered and described in terms of their knowledge representation model and their support for inter-agent communication, and their area of application development. Because of Jason agent language support for logic based representation and inter-agent communication of concepts which is one of the objectives of this research, Jason syntax was analysed in details in its Prolog-like beliefs representation, goals, and plan structures. The chapter introduced CArAgO artifact as a model for developing agent environment interface for observing percepts. The next Chapter 4, presents the PDT AOSE graphical editor tool, chosen because of its detailed engineering process as the software engineering tool for the analysis and design of the Pre-assessment System of this study.

# **Chapter 4**

## **Methodology: Agent Oriented**

## **Analysis & Design and Classification**

## **Method**

### **4. Introduction**

In Chapter 3, the literature of three types of agent oriented methodologies, namely: Gaia, Tropos and Prometheus were presented according to their phase to phase interactive design process. After the analysis of the methodologies, Prometheus was chosen as the agent oriented design approach to apply in this research. This chapter, therefore presents Prometheus in its step-by-step design process for designing agent based system from the initial step of problem description, scenario development, goal specification, agent roles and interaction, protocol analysis and agent capability specification. The chapter then presents the parameters of a student model used in the development of the Pre-assessment System as well the Pre-assessment Mechanism that symbolises the strategy for identifying gaps in students' learning, classifying students and making recommendation for their learning. In addition, the chapter illustrates with examples the modelled rules estimation formula that calculates the number of classification rules for the classifier agent.










### **4.1 Prometheus Agent Oriented Software Engineering**

Agents oriented software engineering (AOSE) is an approach to developing intelligent agent systems. The methodology for analysing, designing and developing a multiagent systems varies. For this research the Prometheus methodology was adopted. The Prometheus method is an approach that engages its graphical editor in engineering the design process. The tool is known as the Prometheus Design Tool (PDT). PDT is an

AUML tool that supports the step-by-step design process. In the following section the range of notation symbols for the interactive design and detailed documentation are introduced.

### 4.1.1 Notation Symbols of PDT

The Figure 4.1 present the PDT notation symbols and their functions in the design of agent based systems.

Name	Symbol	Description
Agent	 Agent	The agent symbol.
Action	 Action	This is what the agent does that has effect on the environment or other agents.
Role	 Role	This symbolises roles or group of roles for agents.
Protocol	 Protocol	Protocols specifies interaction between agents. Protocols are specified using textual notations that maps to AUML2.
Data	 Data	This is used to represent the belief (internal knowledge model) or external data. It is where functionalities that transcends to agent read or write data or information.
Messages	 Message	This is used to symbolise a message communication between agents.
BDI Messages	 BDI Message	This symbol is used to represent messages that updates the beliefs of agents.
Percept	 Percept	Represents the input coming from the environment to the agent.
Scenario	 Scenario	This is an abstract description of a sequence of steps taken in the development of a system. It is usually the initial step that starts for the breakdown of the “statement of problem” or description of the problem to solve.





Goal	 Goal	It is the realisable target or achievement set for an agent.
Connection Arrows	 PDTConnection	They are edges that connects entities (i.e. symbols) together.

Fig.4. 1: PDT notation symbol.

The following section starts the design of the multiagent system for the pre-assessment of students' prior knowledge using the PDT tool. As a set of guidelines, the Prometheus methodology proposes three major agent software development phases, namely: *System Specification*, *Architectural Design* and *Detailed Design*, and PDT supports design through these phases.

## 4.2 System Specification

The specification phase as described in Chapter 3 begins with a high level description of the problem, then the identification of initial goals from the description.

### a) Identifying initial goals:

As stated in Padgham & Winikoff (2004) initial goal specification always begin the process of an entire system goal specification and functioning stages of a multi-agent system (MAS). The following description states and identifies what the system is going to do (Ehimwenma, Beer & Crowther, 2014b; 2015a):

*A student desires to learn a concept. The student enters a concept on the system. The system needs to ensure the student has understanding of prerequisite concepts to the desired concept. The student is tested, learning activities are aggregated and classified in continuous interactive feedback process, and belief store updated all the way. In the end, appropriate learning materials are recommended.*

### b) System goals

Based on the above stated description, the system goals are:

- *Observe percept*

- *Understanding of prerequisite*
- *Testing*
- *Classifying*
- *Continuous feedback*
- *KB update*
- *Recommend materials*

**c) Goal specification**

The question is how can each of these goals be achieved? Each of the goals had further sub-goals developed as follows:

- i) This step is where agent gets percept (e.g. *desired\_concept*) and display it:

**\* *Observe percept***

- Receive user concept
- present concept

**DESIRED\_CONCEPT**

- ii) To the step where quizzes in belief based (BB) are retrieved and presented:

**\* *Understanding of prerequisite***

- quizzes in BB
- answers in BB
- prerequisite assessment from quizzes and answers

**UNDERSTANDING PREREQUISITE**

**NB:** By further re-arrangement or refinement, the sub-goals in the Student has *understanding of prerequisite* goal can become sub-goals of TESTING (below).

iii) This is the step of testing student knowledge:

***\*Testing***

- search BB for quizzes
- fetch (sub-concepts or) prerequisite quizzes
- receive answer
- fetch BB answer and compare with students'
- make assessment decision

**TESTING USER**

iv) To the step where agent gets aggregated BB updates of messages communicated about pre-assessment, matching beliefs in plan context, and classifying student knowledge:

***\*Classifying***

- aggregate learning activity
- use predicate statement rules
- classify students based on rules match

**CLASSIFICATION**

v) To the step where all learning activities are stored persistently:

***\*KB updating***

- store user learning activity persistently

**PERSISTENT BELIEF STORE**

vi) This step shows that the system is continuously interacting and communicating the outcome of every activity to the student:

***\*Continuous user feedback***

- user friendly interaction from assessments
- welcome and introduction to system

**USER INTERACTION**

vii) This is the step where learning materials are recommended for students:

**\*Recommend materials**

- concept ontology in BB
- search ontological relation
- fetch URL link
- present to user

**RECOMMENDATION**

### 4.2.1 Scenario Overview

Scenarios and system goals are complementary. In process of extracting the main goals from the problem description, scenarios were also being developed. The Figure 4.2 shows the set of scenarios derived from the specified goals using the PDT Scenario Overview diagram.

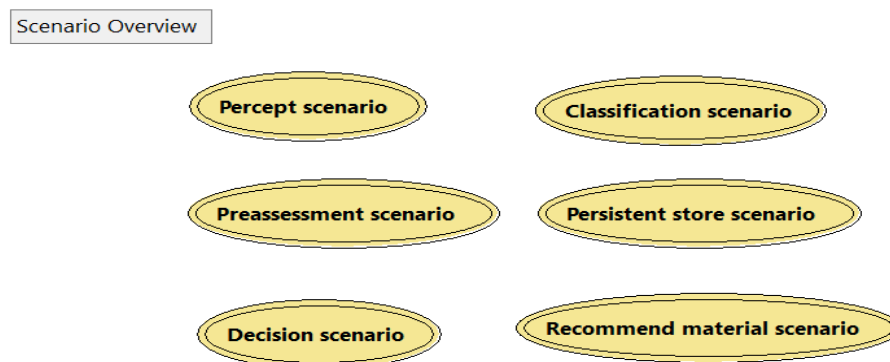


Fig.4. 2: System scenario view.

### 4.2.2 System Goal Diagram

The PDT System goal overview diagram enables the break-down of the set of derived scenarios into units of achievable design steps. The Figure 4.3 is the system goal and subgoals design and the interactions between them. The AND is a conjunction function

which indicates that, at that level of design, the agent must communicate both the *classify* and the *persistentBB update* after its *decision making* function.

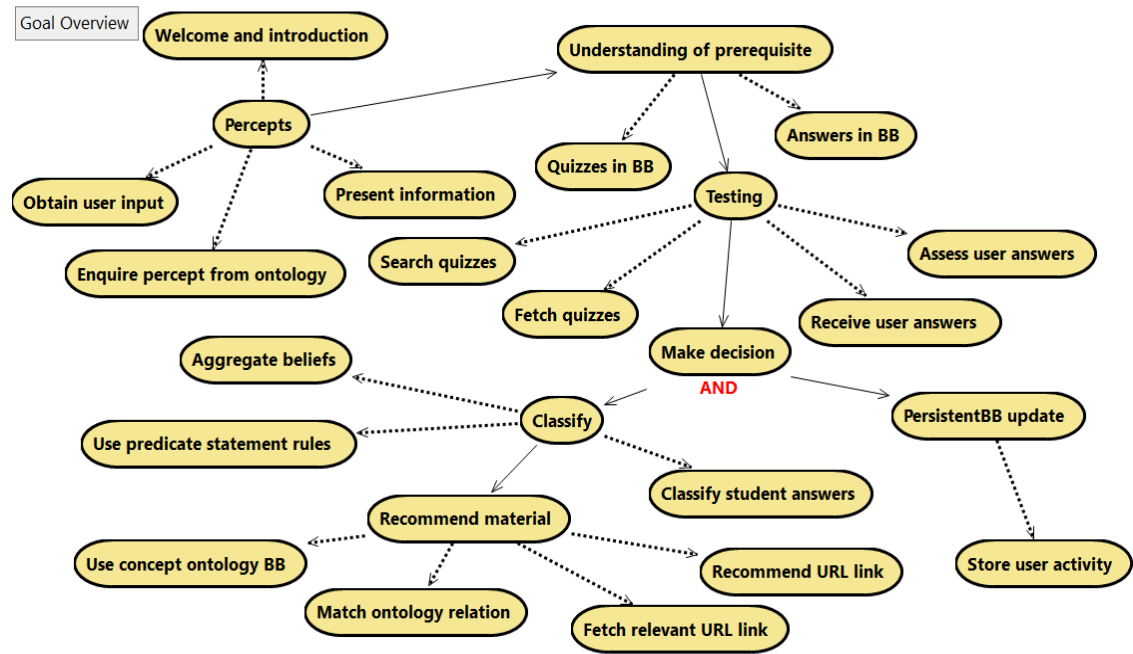


Fig.4. 3: System goals specification for the pre-assessment system.

In the Figure 4.3, the *user interface* goal is seen interacting with the *understanding of prerequisite* goal which connects to the *testing* goal. Then to the *make decision* goal that is linking both the *classify* and *persistentBB update* goals after its decision making function; and the *classify* goal connects the *recommend material* goal. The solid arrow lines are the connections between goals, while the dotted lines are the links between a main goal and its subgoals.

### 4.2.3 Set of Functionalities

From system goals, a set of functionalities are derived as roles for the system. In the step, these roles are grouped together. These roles later turned out to be set of functionalities or roles for the agents.

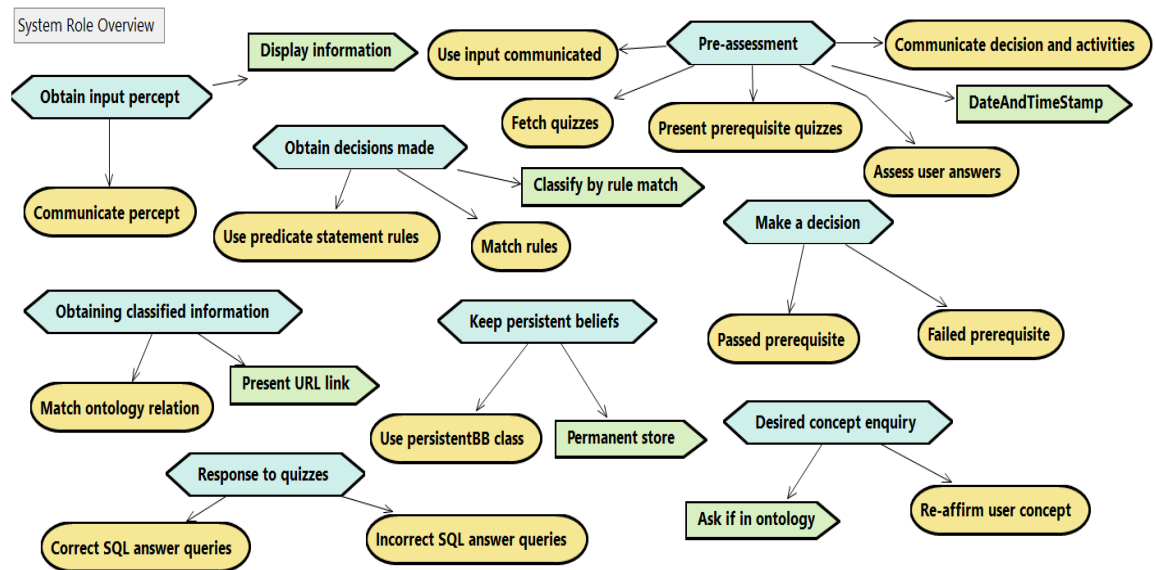


Fig.4. 4: System role overview showing structured Functionalities.

## 4.3 Architectural Design

In this phase, the different agent of the Pre-assessment System has been determined and included in the design. The phase also consists of the system overall (static) structure using system overview diagram, and the description of the dynamic behaviour of the system using interaction diagram and interaction protocols.

### 4.3.1 Analysis Overview

From the system scenario step, interactions within the system is first established using the analysis overview diagram (Figure 4.5). This involved including the agents.

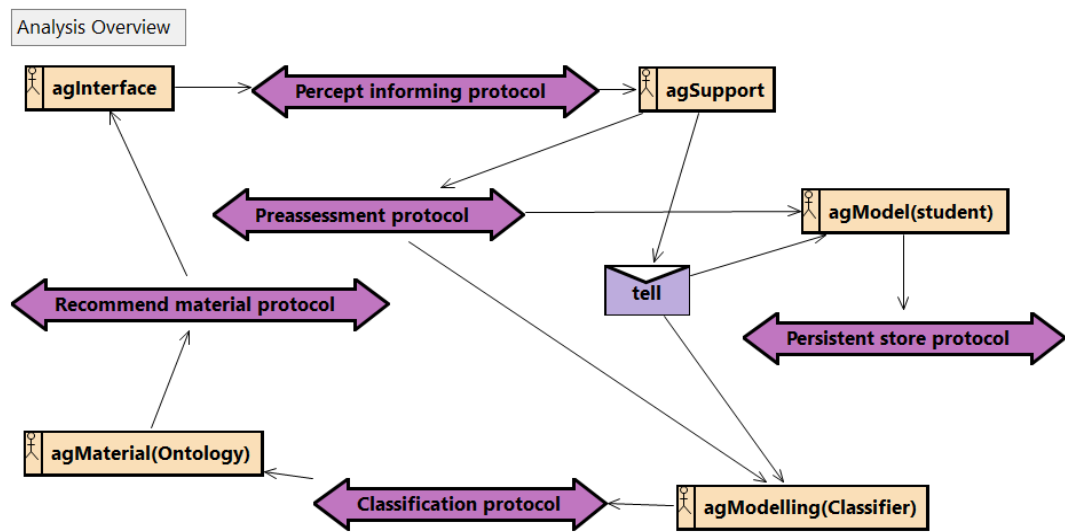


Fig.4. 5: Analysis overview from system scenarios.

### 4.3.2 Agent Role Ordering

Agent roles ordering is the design step for identifying and grouping roles for the respective agents in the system. From the system role grouping of the preceding phase in Figure 4.4, agent roles were ordered in Figure 4.6.

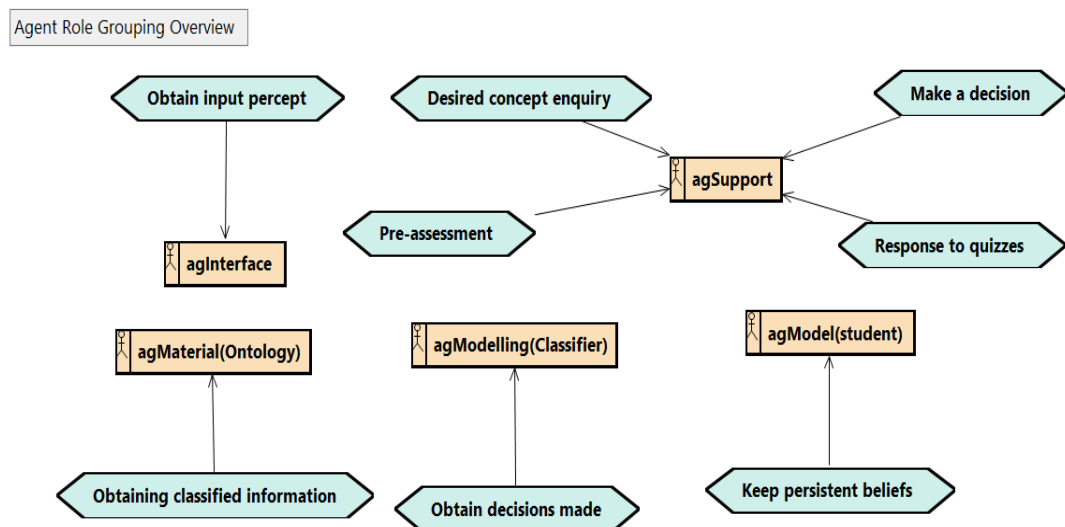


Fig.4. 6: Agent Role Grouping.

### 4.3.3 System Overview

In this step, all the entities, that is, the agents, their percepts, type of messages, actions and interaction in the design (Fig. 4.7). From the System Overview step, protocol interactions between agents were derived using the AUML2 facility (Fig. 4.8). In the system overview diagram, data are also coupled with agents to specify the type of data being used. In this design, the data are quizzes, answers to quizzes, and URL data links for each of the sub-topics (leafnodes) in the ontology. These data are modelled as internal knowledge or beliefs in the agents.

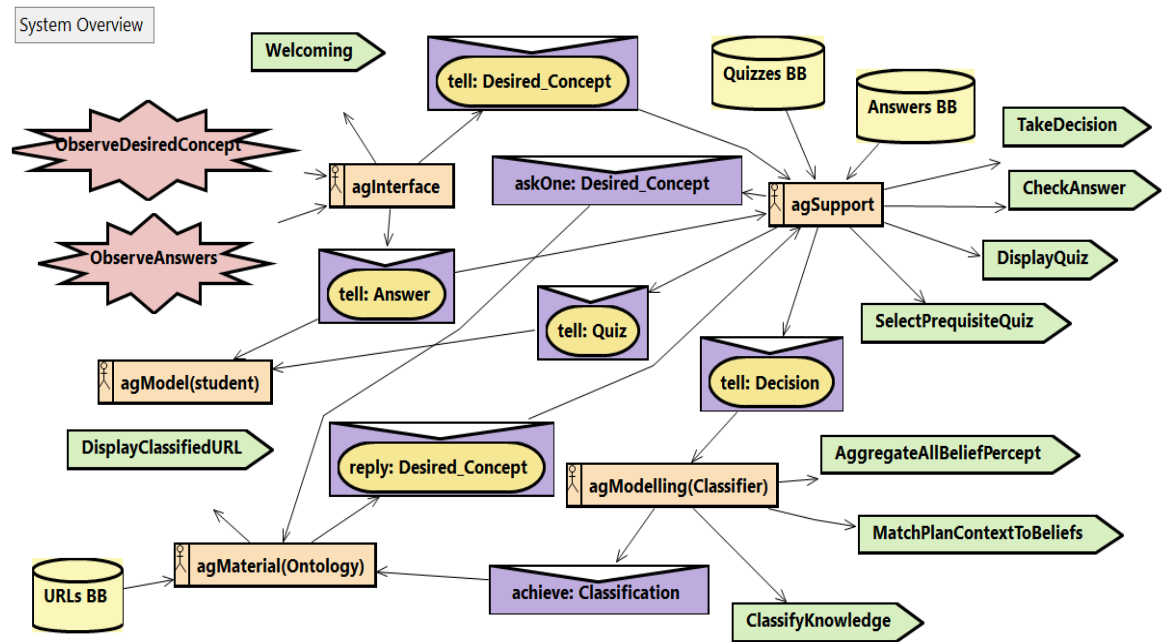


Fig.4. 7: System overview diagram.

To specify protocols interaction design for agents, the AUML commands must be issued. The Figure 4.8 presents the AUML protocol commands that produced the protocol interaction diagram in Figure 4.9 and protocol interaction table in Figure 4.10.



```
start Preassessment process protocol
agent St student
agent T agInterface
agent S agSupport
agent M agModel
agent C agModelling
agent O agMaterial
box alt
  message T St promptDesired_Concept
  message St T Desired_Concept
  message T S tell: Desired_Concept
  message S C tell: Desired_Concept
  message S M tell: Desired_Concept
  message M M permanentStore
end alt
box loop
  message S S fetchPre_Quiz
  message S St displayQuiz
  message St T tell: Answer
  message T S tell: Answer
box alt
guard [Answer OK]
  message S St informPassed
  message S C tell: Passed
  message S M tell: Passed
  message M M storePassed
next
guard else
  message S St informFailed
  message S C tell: Failed
  message S M tell: Failed
  message M M storeFailed
end alt
end loop
box alt
  message C C classify
  message C O achieve: Classification
  message O O fetchMaterialURL
  message O St displayMaterialURL
end alt
finish
```

Fig.4. 8: FIPA-compliant AUML command protocol.

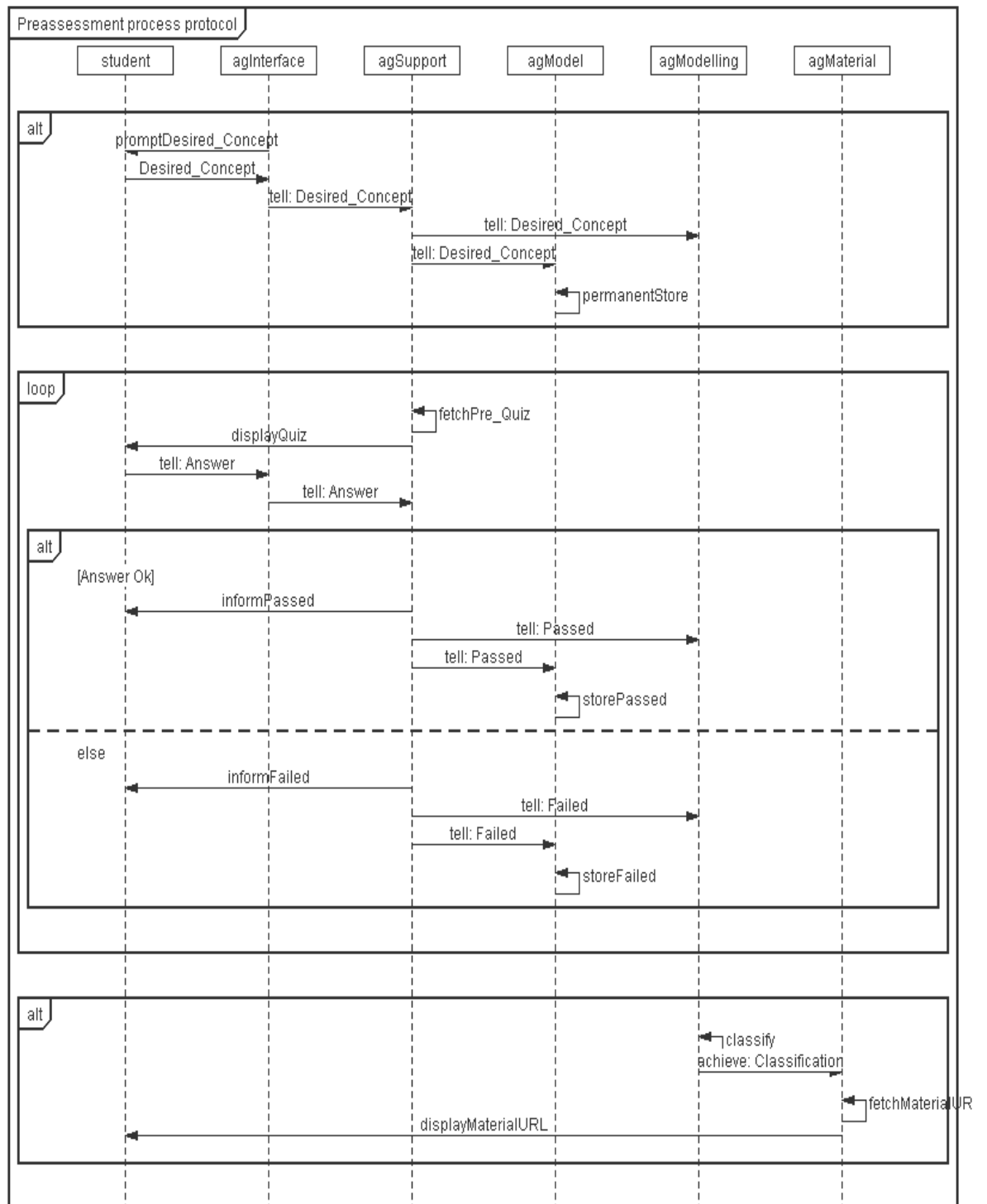


Fig.4. 9: FIPA Compliant AUML protocol diagram analysis for inter-agent interaction. It shows the dynamic interaction of agent message passing via performatives.

The Figure 4.9 has a loop segment. The loop depicts the process where the agent agSupport uses *achievement goals* to navigate from leafnode to leafnode in hierarchy of concepts to retrieve quizzes which are represented as logic formulas in its BB to test students' knowledge.

Interactions Table			
Type	Name	From	To
message	promptDesired_Concept	agInterface	student
message	Desired_Concept	student	agInterface
message	tell: Desired_Concept	agInterface	agSupport
message	tell: Desired_Concept	agSupport	agModelling
message	tell: Desired_Concept	agSupport	agModel
message	permanentStore	agModel	agModel
message	fetchPre_Quiz	agSupport	agSupport
message	displayQuiz	agSupport	student
message	tell: Answer	student	agInterface
message	tell: Answer	agInterface	agSupport
message	informPassed	agSupport	student
message	tell: Passed	agSupport	agModelling
message	tell: Passed	agSupport	agModel
message	storePassed	agModel	agModel
message	informFailed	agSupport	student
message	tell: Failed	agSupport	agModelling
message	tell: Failed	agSupport	agModel
message	storeFailed	agModel	agModel
message	classify	agModelling	agModelling
message	achieve: Classification	agModelling	agMaterial
message	fetchMaterialURL	agMaterial	agMaterial
message	displayMaterialURL	agMaterial	student

Fig.4. 10: AUML Protocol Interaction table.

## 4.4 Detailed Design

This phase is focused on the description of responsibilities and capabilities of the internal structure of the individual agent, and how they will achieve their task within the system. Diagrammatically, these capabilities have been realised on the agent overview canvass.

### 4.4.1 Agent Overview

In this section, individual agent internal details are presented. Using the plan notation symbol, percept, triggering event, inter-agent messages and data are specified. At the

agent overview stage, inherited interfaces from e.g. the system overview phase are adopted for specifying agents' details. The inherited interfaces, that is, notation symbols are those that appears greyish in colour.

a) Agent *agInterface*

In Figure 4.11 is a much refined detailed design where CArtAgO artifact is the medium to get input from the user is specified.

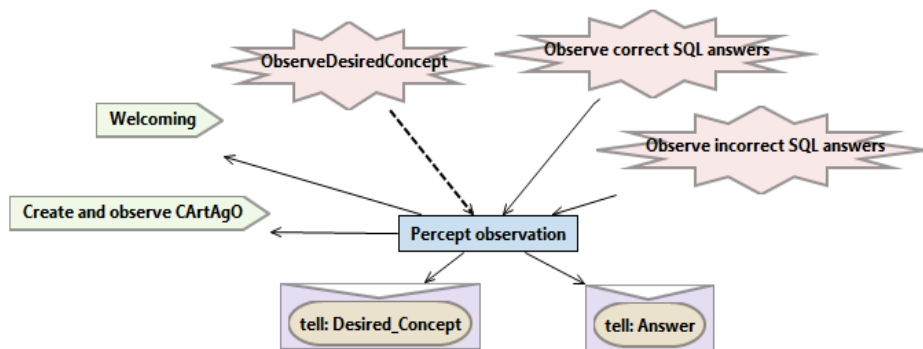


Fig.4. 11: Detailed overview of agent *agInterface*.

The interface agent first creates the artifact in order to observe it. All inputs that are observe are communicated as messages, in agent plan (shown with the plan diagram or symbol), to the agent *agSupport* that is responsible for pre-assessing students.

b) Agent *agSupport*

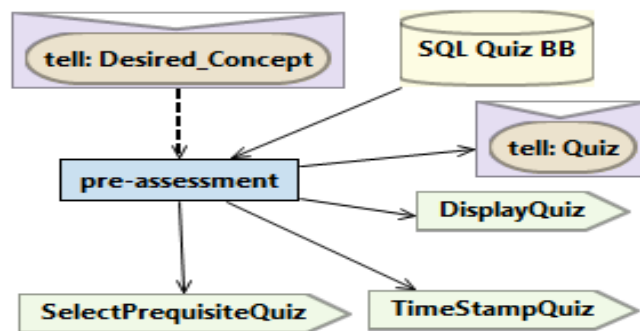


Fig. 4. 12: Agent *agSupport* receiving the *desired\_Concept* percept and retrieving quizzes.

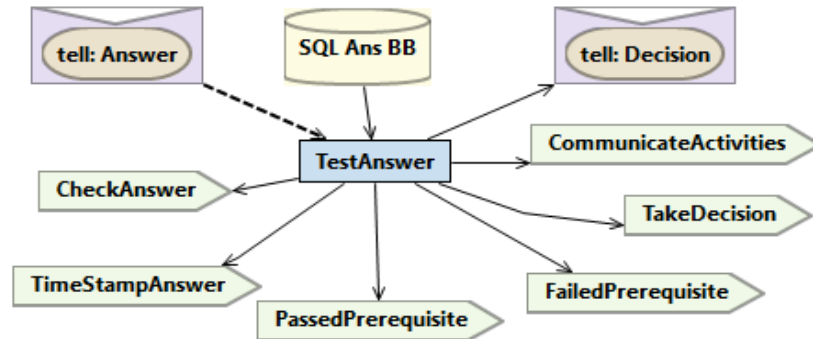


Fig.4. 13: Agent *agSupport* Overview: Using answer percept to make comparison. Taking pass or a fail decision, and communicating all activities and decision reached to other agents of the MAS by its agent plans. This agent also date and timestamp learning activities.

### c) Agent *agModelling*

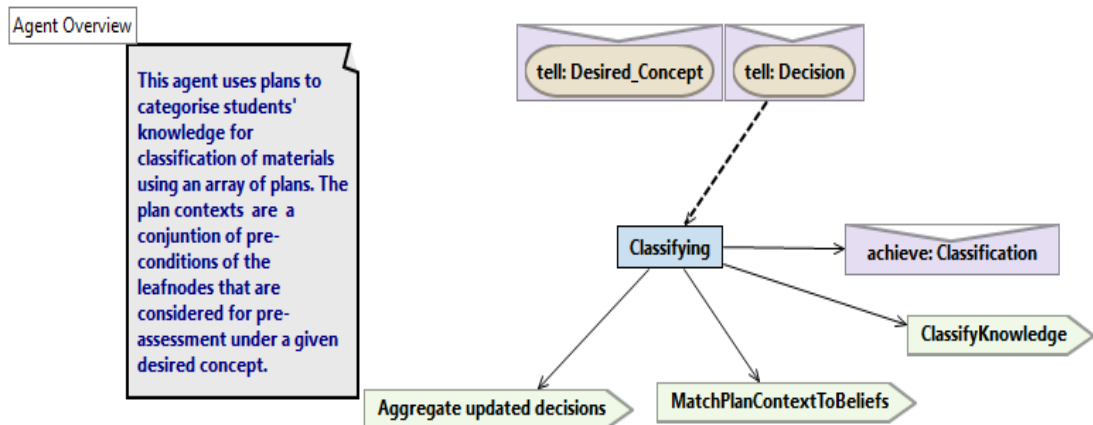


Fig.4. 14: The agent *agModelling*: The classifier agent Overview

This agent gets message percepts from agent *agSupport* for every leafnode whose pre-assessment is completed. It starts matching the right pre-conditions in plan *context* with the messages received, and thereafter select the appropriate categorisation of students.

d) Agent agMaterial

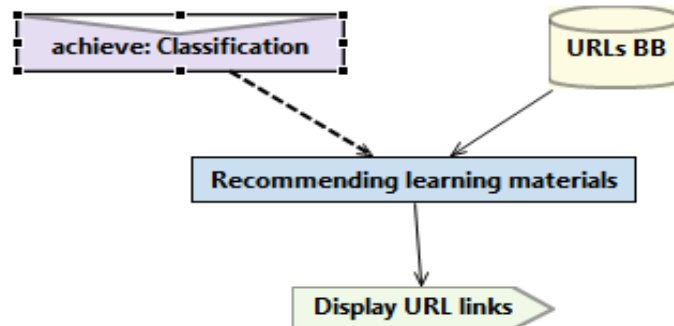


Fig.4. 15: Agent *agMaterial*: The learning material agent Overview.

This is agent *agMaterial* keeps the URLs links of learning material as ontology. At the receipt of an *achieve* performative message from the classifier agent (after classification), the agent *agMaterial* then releases learning materials for students to learn. These materials are dependent on the number of *failed* and *passed* prerequisite assessment.

e) Agent agModel

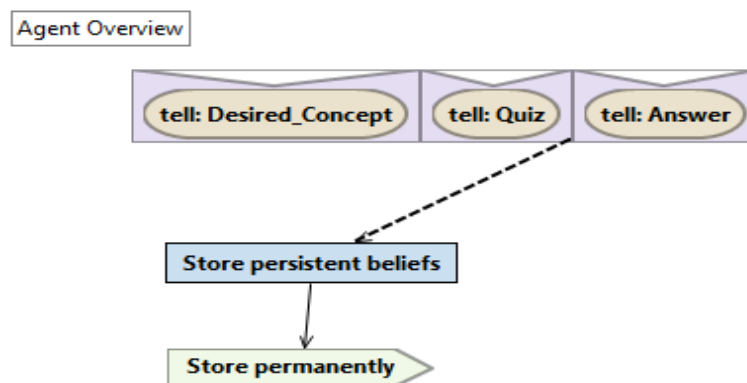


Fig. 4. 16: Agent *agModel* (student) Overview

This agent uses the Java *TextPersistentBB* class to store all the learning activities in the system. The *TextPersistentBB* is configured in the MAS at the point of declaring or naming the agents *.Mas2j* project level of implementation. The activities stored are

messages to the agent, and they are desired concepts, answers (both correct or incorrect) percept. This plan keeps other information such as *desired\_Concepts*, and quizzes apart from the SQL answer queries from students.

#### 4.4.2 Roles and Capability Descriptors for Agents

In summary, the Figures 4.17 and 4.18 outlines the detailed Capability Descriptors of the agents in the system. While Roles are the functionalities meant for agents to achieve, Capabilities are a set of related plans used for realising goals. Goals are steps through which agent fulfill their intentions.

Roles	Goals	Capability/plan
Obtain input percept	-Communicate percept -Display percept	Capability
Pre-assessment	-Use input communicated -Percept request from ontology -Present prerequisite quizzes -Compare answer percept with BB -Take decisions -Communicate decisions and activities -Date and timestamp activities	Capability
Obtain decisions made	-Aggregate updated decisions -Use predicate statement rules -Match rules -Classify by rule match	Capability
Obtaining classified information	-Search ontology BB -Match URL ontological relations -Present URL link	capability
Keep persistent information	-Use persistentBB class -Store persistently	Capability

Fig.4. 17: Capability descriptor.

Goals	Plans	Actions	Percepts	Internal Action	Data
Communicate percept	In a plan	Performatives: tell, achieve	Triggering event: desired_Concept, SQL answer queries	.send	N/A

**Chapter 4** *Methodology: Agent Oriented Analysis & Design and Classification Method*

Display percept	In a plan	Screen print	Triggering event: desired_Concept	.print	N/A
Percept request from ontology	In a plan	askOne request	Triggering event: desired_Concept		Ontology BB
Use input communicated	In a plan		Triggering event: desired_Concepts, correct SQL answers, incorrect SQL answers		N/A
Present prerequisite quizzes	In a plan	Goals, subgoals, and screen print	Triggering event: desired_Concept, SQL answer queries (correct/incorrect)	.print	Quizzes BB, Answers BB
Compare answer	In a plan	Feedback to student: pass or fail	Triggering event: SQL answer queries (correct/incorrect)		Quizzes BB, Answers BB
Take decisions	In a plan	Make a pass or a fail decision	N/A		N/A
Communicate decisions and activities	In a plan	Send answers logged in by students, [passed or failed] predicate messages	N/A	.send	N/A
Aggregate updated decisions		Update beliefs with all the decisions [Passed or Failed] received	Passed or Failed prerequisite decisions		N/A
Match rules	Set of plans	Match plan context with updated beliefs	Triggering event: desired_Concept, SQL answer queries (correct/incorrect)		N/A
Classify by rule match	By a plan	Select the relevant plan and communicate recommendation message	N/A		N/A
Match URL ontology relations	In a plan	Match or unify plan context	Triggering event: Recommendation message		N/A
Present URL link	In a plan	Release URL link	N/A	.print	N/A
Store persistently		Use persistentBB class	Triggering event: desired_Concept, SQL answer queries (correct/incorrect)		Text Persistent BB
<b>Goals</b>	<b>Plans</b>	<b>Actions</b>	<b>Percept</b>	<b>Internal Action</b>	<b>Data</b>

**Fig.4. 18:** Expanded summary of capability descriptor: percepts, triggering events, goals, plans and data used by agents in the system.



## 4.5 The Student Model

Baffes (1994) states that a *student model* involves the method used in representing the knowledge of students. As given in Padayachee (2002), modelling a system for learning purposes involves the use of interactive component and attributes of the learner (i.e, the student). The Classical Four Model (Padayachee, 2002) architecture as shown in Chapter 2 has a Tutoring Module that uses: a strategy for *diagnosing* misconception and learner's need, a *module* that stores a student's current cognitive status, a *knowledge base module* containing domain knowledge and the procedure of learning, and a *user interface* for interactive dialog. The agent based Pre-assessment System of this study mirrors this type of ITS architecture where a diagnostic strategy is being employed to identify gaps in students' learning in a system that can also collect students' learning activities, keep students' learning attributes and classify students' knowledge for learning materials.

Agents are designed to observe their environment. The environment to observe in this research are not natural environments. Rather a student environment that is part of a software system (Ricci, Piunti, Viroli, 2011). Wang (2014) called this environment a *partially observable* environment. In this research, for agents to observe the student environment, the environment needs to be modelled with the parameters that can elicit and represent the inherent knowledge attributes of students with regards to identifying gaps in their learning. To this effect, a student model was devised with five parameter information from the viewpoint of the Tutoring Module (Padayachee, 2002) that can diagnose misconception in students' learning. In a tuple, the model is given as:  $M = \langle D, C, P, F, V, S \rangle$  (Ehimwenma, Beer & Crowther, 2015a; 2015b) where

- $\langle M \rangle$ : is the model.
- $\langle D \rangle$ : The *desired\_Concept* is the set  $D = \{C_1, C_2, \dots, C_{k-1}, C_k\}$  of observable parent classes in an ontology tree that has leafnodes  $N$  such that  $N_{i,j}$  are the set of leafnodes with respect to  $C_i$ .
- $\langle C \rangle$ : The set of *prerequisite* such as  $C = \{C_2\}$ ;  $C = \{C_2, C_3\}$ ; or  $C = \{C_2, C_3, \dots, C_{k-1}, C_k\}$  parent classes underneath a *desired\_Concept*  $D$ . In general, a

*prerequisite* to a *desired\_Concept*  $C_i$  is  $C_i - C_{i-1}$ . For instance, let  $C_1$  be a *desired\_Concept*, then any other element of the set  $C$  can be a *prerequisite(s)* to  $C_1$ , respectively. That is, a  $D \equiv C$ .

- **<P>**: The set of **passed** predicate  $P = \{p_1, p_2, \dots, p_{k-1}, p_k\}$  over the leafnodes  $N$  of the *prerequisites*  $C$  to a *desired\_Concept*  $D$ . The first order logic (FOL) form is  $P(N_{i,j})$  for a given leafnode. Thus, for the *prerequisite*  $C$ , the index  $x$  in  $N_x$  represents the total number of individual leafnode  $N$  per  $C_i$ . Therefore,  $N \sqsubseteq C$  *i.e.*  $N$  is subclassed by  $C$ , and  $C \sqsubseteq D$  *i.e.*  $C$  is subclassed by  $D$ . At start of pre-assessment any  $D \equiv C$ . The  $P(N_{i,j})$  formula symbolises knowledge gain.
- **<F>**: The set of **failed** predicate  $F = \{f_1, f_2, \dots, f_{k-1}, f_k\}$  over the leafnodes  $N$  of the *prerequisites*  $C$  with respect to a *desired\_Concept*  $D$ . In FOL formula this is given as  $F(N_{i,j})$  for a given leafnode  $N$  per  $C_i$ . The  $F(N_{i,j})$  formula symbolises knowledge gap.
- **<V>**: The set of **observable inputs** e.g. SQL answer queries  $V = \{V_1, V_2, \dots, V_{k-1}, V_k\}$  from students over the leafnodes  $N$  of the *prerequisite*  $C$  to a *desired\_Concept*  $D$ . For every correct answer input that is assessed, the atomic formula  $P(N_{i,j})$  as the corresponding decision statement is taken and communicated; for every incorrect answer input, the corresponding predicate  $F(N_{i,j})$  decision statement is taken and communicated for appropriate classification.
- **<S>**: The set of **timespent**  $S = \{s_1, s_2, \dots, s_{k-1}, s_k\}$  by a student on pre-assessment activities; such that,  $s_k$  is the time interval between a given question on the system and the student answer. This is so because every activity and the expected students' response are timestamped by an agent.

The choice of the parameters **<D>**, **<P>** and **<F>** which are predicates for first-order logic statements, a form of knowledge representation stated in Chapter 2 (e.g. *Father(peter)* (Baader & Nutts, 2003)). In addition, the **<D>**, **<P>** and **<F>** are for agents' communication and for *reasoning* by the agent *agModelling* for the categorisation of students for learning materials. This is in contrast to SmartTutor (Gamalel-Din, 2002) where *learning-by-experience* was used. The use of these

parameters in this research is informed by their absence in literature as predicates in logic based statements for multiagent systems development.

The  $\langle P \rangle$  and  $\langle F \rangle$  represents the predicates for the logic based decisions statements in the agent *agSupport* plan after every pre-assessment. They represent boolean values. While the  $\langle P \rangle$  is the predicate in the logic statement that will communicate the decision on correct answer response, the  $\langle F \rangle$  is the predicate that would communicate the decision on the incorrect answer response. From the model M, above, the following outlines the purpose of the modelled parameters in the Pre-assessment System:

- ***To fetch and communicate observed percepts (inputs) from the environment:*** Consider  $\langle D \rangle$  or *desired\_Concept* as any topic or concept a human tutor, for instance, wants to teach. The Pre-assessment System, like the tutor wants to know whether students are prepared for  $\langle D \rangle$ . Then the system pre-assesses students on the past prerequisites  $\langle C \rangle$ . To fetch quizzes of prerequisite concepts, agent uses `!achievement goals`.
- ***To construct classification rules for agent:*** To classify students for appropriate learning material, the classifier agent *agModelling* gets messages from the pre-assessment agent *agSupport* with a *tell performative*. This messages are the decisions reached after each pre-assessment. The decisions statements that are communicated are logic based formulas with  $\langle P \rangle$  and  $\langle F \rangle$  as predicates. After aggregating the messages, the plan context that is matched in the agent *agModelling* would be triggered, and further message communication is sent using the `achieve` performative to agent *agMaterial* (Fig. 4.14).
- ***To support the release of URL links after classification:*** The message expected by the agent *agMaterial* are recommendation triggers from agent *agModelling*. When the agent *agMaterial* gets these messages, it also matches the appropriate plan context and release the URL(s) for learning material(s) (Fig. 4.15).

- **To keep student learning history:** In order for the tutor to unravel possible difficulties facing his students in the domain context (i.e. SQL) of learning (of this research), the *TextPersistentBB* class shall be configured in the MAS for the agent *agModel* to keep the students' learning history persistently. These are information that includes: the <D>, <P>, <F>, and <V> attributes. The <V> parameter are answers to be viewed by the tutor to support students in SQL. The *TextPersistentBB* is a Jason *TextPersistentBB* class (a text database) (Fig. 4.16).

In addition, the parameters <P> *passed* or <F> *failed* are not chosen nor devised for first-order logic statements for classification alone. But also to reinforce students (e.g. Pavlov, 1960) in the course of pre-learning assessments.

## 4.6 The Pre-assessment Mechanism

The pre-assessment mechanism is a structure devised to present the picture of the process of identifying gaps in students' learning and making supplementary learning materials recommendation. The function is to ascertain the true and accurate level of students' skills and knowledge and supporting them to start learning at the level appropriate to their current level of knowledge because every student cannot afford to start from the same learning block. This approach is similar to the PAT (Ritter *et al.* 1998) strategy that ensures that current skills set for students are attained before promoting students to a new level of learning.

This structure (Fig. 4.19) depicts:

- How learning concepts are represented in hierarchy.
- The strategy for decision flow and navigation from leafnode concept to leafnode concept for prerequisite question selection when a *desired concept* is received; which would be released by the use of agent *achievement goals* (Bordini, Hubner & Wooldridge, 2007).
- The communication of the decisions made within the system after every pre-assessment.

- The aggregation of decision statement.
- The classification of students learning using the aggregated decision statements for learning materials recommendation.

In the *Pre-assessment Mechanism* (Fig. 4.19) learning concepts are given in a hierarchy of inter-related concepts illustrated with the letters *A*, *B*, *C*, and *D*. Where *A* represents the lowest class concept and *D* the highest class concept in a hierarchy of learning structure. The *A*, *B*, *C*, and *D* represents any class nodes or topics in the SQL domain of learning. Every class node has at least two leafnodes and a subclass node that has its own leafnodes. The leafnodes are the concepts that represents the lessons taught in the classroom.

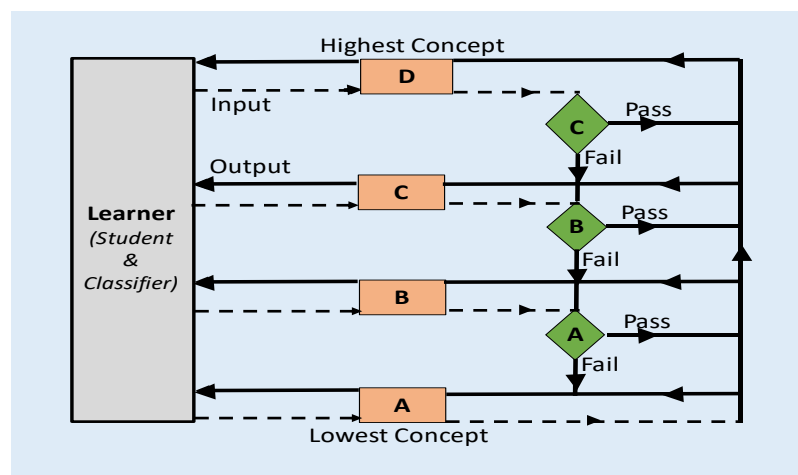


Fig.4. 19: The Pre-assessment Mechanism (Ehimwenma, Beer & Crowther (2014b))

## 4.7 The Learner Component

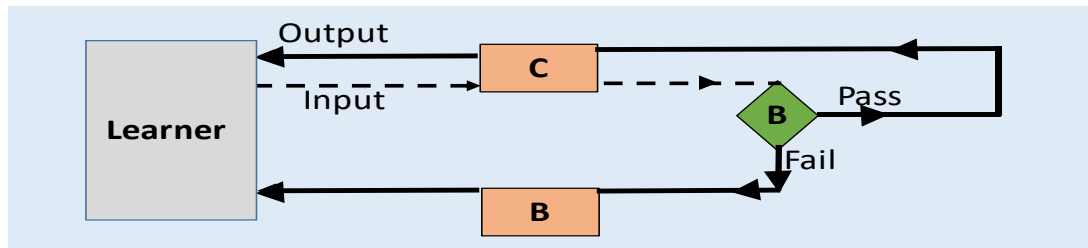
The Learner component in the Pre-assessment Mechanism is dual purpose: i) as students and ii) as a classifier. The first input into the system by students are the desired concepts as symbolised with *A*, *B*, *C* or *D* in the Figure 4.19. Where *A* is the bottom (or lowest concept) that has no prerequisite. As such *A* has no pre-assessment and becomes the default concept to study when entered.

When a student enters a class node (i.e. *desired\_Concept*), agent !achievement goal is triggered to retrieve the quiz corresponding to a leafnode of the prerequisite class, then pre-assessment is carried out, decision is taken based on the answers

received; and then followed by the next !achievement goal according to the number of leafnodes considered under the *desired\_Concept* (see Fig. 4.9 for the loop in the PDT AUML protocol diagram). As shown in Figure 4.19, a *pass* or a *fail* decision is taken by the MAS for every quiz that is completed. While the student is getting *feedback* about his/her performance, the beliefs of the *classifier* agent is also being updated with the *pass* or *fail* decisions to match the relevant plan *context*, and the student is classified for learning material(s). Thus, because of the need of a system to gather students' skills status (or decisions), classify them and make recommendation for learning materials, a multi-agent system was considered as appropriate to provide this capability. This is due to the fact that individual agent can handle specialised functions. Case based reasoning (CBR) is a type of classification technique that was combined with MAS in González, Burguillo & Llamas (2005). CBR is a method in which concrete previous experience is applied to solve current and similar problem situations. In contrast to CBR approaches where a current problem is interpreted as a previous one based on similarities or differences (classification CBR), or where a new solution is adapted based on past, stored or existing solutions (problem CBR) (de Mantaras, 2001); the approach taken in this thesis is a rule-based approach to reasoning by a *classifier* agent. This is where domain specific rules are specified as antecedents for a body of conclusions that is applied in a classification process (Patterson, 1990, Rifkin & Klautau, 2004; Marsland, 2014). This is because, we believe that the rule-based approach is more decisive to address the errors that are liable to be made by students in their responses to questions from the system that will in the end make recommendation for their learning. In addition, because the answer input to the system is open ended, so answers submitted by students to the system may also not be similar. In this process, all pre-assessment activities will be communicated between agents as specified in the PDT diagrams (e.g. Fig. 4.9). This process of pre-assessment as regards the Pre-assessment Mechanism (Fig. 4.19) can be viewed in two ways for implementation, namely: i) Pre-assessment by immediate prerequisite class, and ii) Pre-assessment by multiple prerequisite classes.

## 4.8 Pre-assessment By Immediate Next Prerequisite Class

This is the pre-assessment strategy that considers only the leafnodes of the immediate prerequisite class to a *desired concept* that is intended for learning by a student (Fig. 4.20).



**Fig.4. 20:** Strategic diagram of the Pre-assessment by immediate next prerequisite class. Where C represents the desired amongst the classes of concept and B the immediate prerequisite class to C.

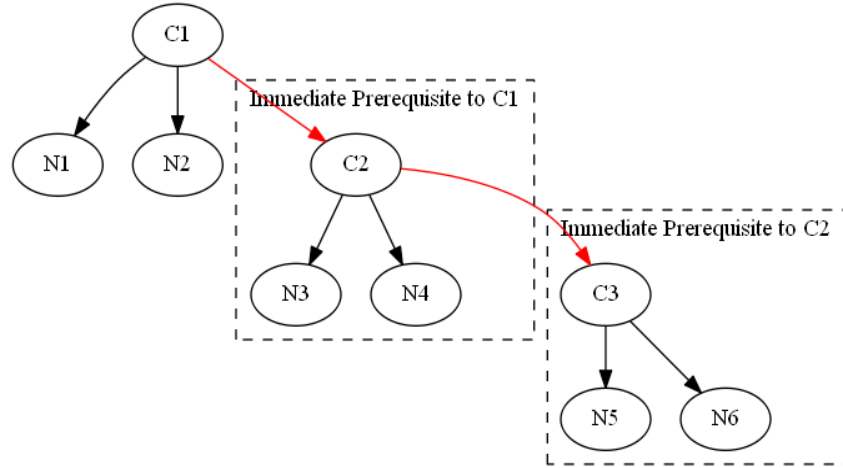
The strategy of the testing process has been shown in the loop segment of the AURL protocol and interaction diagram (Fig. 4.9), and detailed process of pre-assessment rules formation is given in the following section using the Figure 4.21 for illustration. The rule formation procedure is in logic based semantics. As mentioned in Chapter 2, it is described in Dell'Acqua *et al.* (1999) as the use of symbolic representations in the expression of rules, reasoning and knowledge preferences that reacts to several alternative choices of action.

### 4.8.1 Logic Based Classification Specification for Pre-assessment in a Regular Ontology Model

The Figure 4.21 is an ontology tree structure of equal number of *leafnodes*  $N_x$  per parent class node ( $C_i$ ). The tree is a directed graph that shows the relations between a parent class and its subclasses. Furthermore, it illustrates the process of navigation between classes. For instance, let us choose  $C_2$  to be a  $C_i$  then its means for its  $N_{i,j}$ :  $N_3$  corresponds to  $N_{2,1}$ ; and  $N_4$  to  $N_{2,2}$

Now, given that  $C_1$  is a *desired concept*, a pre-assessment would be on the *leafnodes*  $N_3$  and  $N_4$ ; and for  $C_2$  as a *desired concept*, pre-assessment would be on *leafnodes*  $N_5$  and  $N_6$ . In the case where  $C_1$  is the *desired concept*, and *leafnodes*  $N_3$  and  $N_4$  are *passed*, the student learns the *leafnodes*  $N_1$  and  $N_2$  which are *leafnodes* (or childnodes) of the

*desired concept*. Otherwise, the *failed leafnodes*  $N_3$  or  $N_4$  or both are learned. In the case where  $C_2$  is the *desired concept*, and *leafnodes*  $N_5$  and  $N_6$  are *passed*, the student learns the *leafnodes*  $N_3$  and  $N_4$  which are *leafnodes* (or *childnodes*) of the *desired concept*  $C_2$ . Otherwise, the *failed leafnodes*  $N_5$  or  $N_6$  or both are learned.



**Fig.4. 21:** A digraph of a regular ontology tree.

Applying first order logic (FOL) formulas, the classification and recommendation rules for the *classifier* agent to classify students for learning are as stated:

$$\forall \text{desiredConcept}(C_1) \forall N_3 \forall N_4$$

$$[$$

$$: \exists \text{desiredConcept}(C_1) \wedge \exists \text{passed}(N_3) \wedge \exists \text{passed}(N_4) \Rightarrow \text{desiredConcept}(C_1).\{N_1, N_2\} . \quad (1)$$

$$: \exists \text{desiredConcept}(C_1) \wedge \exists \text{passed}(N_3) \wedge \exists \text{failed}(N_4) \Rightarrow \text{failed}(N_4) . \quad (2)$$

$$: \exists \text{desiredConcept}(C_1) \wedge \exists \text{failed}(N_3) \wedge \exists \text{passed}(N_4) \Rightarrow \text{failed}(N_3) . \quad (3)$$

$$: \exists \text{desiredConcept}(C_1) \wedge \exists \text{failed}(N_3) \wedge \exists \text{failed}(N_4) \Rightarrow \text{failed}((N_3) \wedge (N_4)) . \quad (4)$$

]

$$\forall \text{desiredConcept}(C_2) \forall N_5 \forall N_6$$

$$[$$

$$: \exists \text{desiredConcept}(C_2) \wedge \exists \text{passed}(N_5) \wedge \exists \text{passed}(N_6) \Rightarrow \text{desiredConcept}(C_2).\{N_3, N_4\} \quad (5)$$

$$: \exists \text{desiredConcept}(C_2) \wedge \exists \text{passed}(N_5) \wedge \exists \text{failed}(N_6) \Rightarrow \text{failed}(N_6) . \quad (6)$$

$$: \exists \text{desiredConcept}(C_2) \wedge \exists \text{failed}(N_5) \wedge \exists \text{passed}(N_6) \Rightarrow \text{failed}(N_5) . \quad (7)$$

$$: \exists \text{desiredConcept}(C_2) \wedge \exists \text{failed}(N_5) \wedge \exists \text{failed}(N_6) \Rightarrow \text{failed}((N_5) \wedge (N_6)) . \quad (8)$$

]



The  $N_{i,j}$  in the  $passed(N_{i,j})$  and  $failed(N_{i,j})$  logic based notation are decision statements about a student's performance on the ontology leafnodes after pre-assessment on that given node  $N_{i,j}$ . The stated axioms are rules-based reasoning where each axiom represents a case or a category in the pre-assessment of the leafnodes  $N_3$  and  $N_4$ , and  $N_5$  and  $N_6$ , respectively, before a student learns a *desired concept*. The rules which are 8 in number defines the condition for the pre-assessment of immediate prerequisite leafnodes, and also presents the rule structure for a two leafnode per class node in a regular ontology as shown in Figure 4.21. Each rule is a parameter combination of the  $\langle P \rangle$  and  $\langle F \rangle$  predicates in combination with the *desired concept*  $\langle D \rangle$ . The  $\langle D \rangle$  parameter represents the concept entered by a student which is also part of the conditions in the *classifier* agent plan context as implemented in Chapter 5.

Rule (1), for instance,

$$\forall \text{desiredConcept}(C_1) \forall N_3 \forall N_4 : \exists \text{desiredConcept}(C_1) \wedge \exists \text{passed}(N_3) \wedge \exists \text{passed}(N_4) \\ \Rightarrow \text{desiredConcept}(C_1).\{N_1, N_2\}$$

depicts that for all  $\forall$  *desired concept* that is  $C_1$ , for all leafnode  $N_3$ , and for all leafnode  $N_4$ , such that, there exists  $\exists$  in the agent beliefs the *desired concept*  $C_1$  and there exists a *passed* pre-assessment of the leafnode  $N_3$  and there exists a *passed* pre-assessment of the leafnode  $N_4$ , then the conclusion and recommendation for learning shall be the leafnode  $N_1$  and  $N_2$  of the *desired concept*  $C_1$  which is the intended concept of learning submitted by the student. This rule formation system also applies to the class node  $C_1$  whose pre-assessment would be on the leafnodes  $N_5$  and  $N_6$ .

In the Figure 4.21 tree structure, there are four rule axioms per parent class node *if and only if* the immediate class prerequisite to a *desired concept* is considered for pre-assessment. This type of strategy implements *Chunking* (Casteel, 1988; Anderson, 2008) that was discussed in Chapter 2 as the breaking down of skills and learning materials into smaller and more manageable units for students to succeed.

Knowing the number of expected classification rules prior to coding as observed in this work is crucial so as to avoid misclassification or missing out a case of

classification. To estimate the number of expected rules needed, Ehimwenma, Beer & Crowther (2015a; 2015b) devised the *Initialisation* equation:

$$R = CT^N + I$$

Systematically, in navigating from one parent class node  $C$  to another and to their respective leafnodes  $N$ , the classified rules estimation process is expressed as

$$R = C_i T^{N_{i,j}} + I$$

where

$C_i$  = number of prerequisite classes

T = the Boolean parameters <P> and <F> which equals 2

$N_{i,j}$  = leafnodes with respect to class  $C_i$

In a *regular* ontology where pre-assessment is on the immediate prerequisite to a parent class node, the total number of rules  $R$  can be estimated such as illustrated with the Figure 4.21. Given that the total prerequisite class node  $C = 2$  (i.e.  $C_2$  and  $C_3$  in Fig. 4.21), and size of leafnode  $N = 2$  across each parent class, then

$$R = 2 * 2^{**}2 + 1$$

$$R = 2 * 4 + 1$$

$$R = 8 + 1$$

$$R = 9$$

Where  $I$  represents the default rule that corresponds to the lowest concept  $A$  in the Pre-assessment Mechanism that has no prerequisite, as mentioned earlier. The default rule represents the release of the URL link of the lowest concept when entered.

Alternatively, our pre-assessment rules *polynomial* equation (Ehimwenma, Crowther & Beer, 2016b):

$$R = I + \sum_{i=1, j=1}^k C_i T^{N_{i,j}}$$

also estimates the accurate number of rules for the aforementioned regular ontology such that each prerequisite class node  $C_i$  (i.e.  $C_2$  and  $C_3$ ) upon which the pre-assessment will be done takes a unit value of 1, the  $N_{i,j}$  per  $C_i = 2$ ; and T = 2 (the *passed* and *failed* predicates). Thus, by isolating the node and then the summation, we have

$$R = 1 + \Sigma[[C_2T^{N_{2,1}}, C_2T^{N_{2,2}}], [C_3T^{N_{3,1}}, C_3T^{N_{3,2}}]]$$

$$R = 1 + C_2T^2 + C_3T^2$$

$$R = 1 + (1 * 2 ** 2) + (1 * 2 ** 2)$$

$$R = 1 + 4 + 4$$

$$R = 9$$

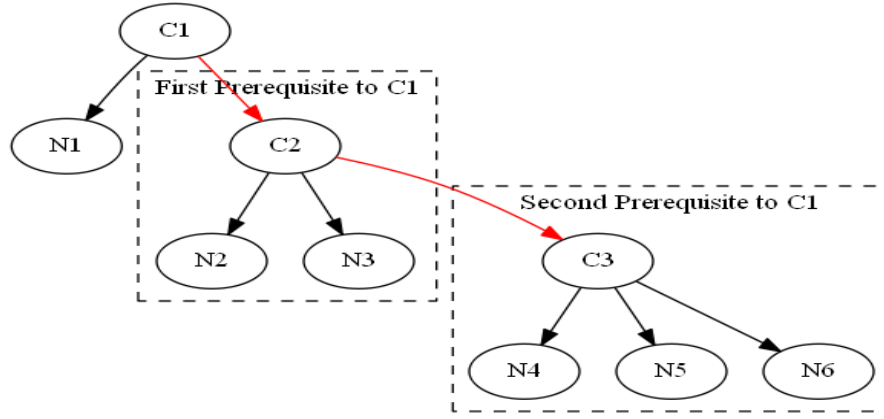
But the estimation of the expected number of rules and the corresponding number of classification rules representation is however different when pre-assessment is of multiple classes beneath a given *desired concept* as shown in the following section.

## **4.9 Pre-assessment By Multiple Prerequisite Classes**

This is the strategy where pre-assessment is from *prerequisite* class to *prerequisite* class under a *desired concept*. In this type of arrangement, the more the number of leafnodes under a given *desired concept*, the more the complexity in the rule representation process. This complexity extends to students in managing their learning gaps having to deal with large amount of recommended URL links, particularly when there is large amount of incorrect responses to pre-assessment quizzes. The loop segment of the AURL protocol and interaction diagram (Fig. 4.9) also depicts this strategic process of pre-assessment and does not specify any size. The Figure 4.22 is *non-regular* ontology that is used to illustrate the rule formation process of ontology of 5 leafnodes.

### **4.9.1 Logic Based Classification Specification for Pre-assessment in a Non-Regular Ontology Model**

The Figure 4.22 is non-regular ontology tree. As against a regular ontology tree that has equal number of leafnodes  $N_x$  across all parent class  $C_i$ , a non-regular ontology is a tree with a varying of number of leafnodes across its parent class  $C_i$  node.



**Fig.4. 22:** A digraph of non-regular ontology tree. A model where all the prerequisite classes under a given parent class, in this case  $C_1$ , are being considered for pre-assessment.

The parent classes  $C_i$  in the tree (Figure 4.22) are  $C_1$ ,  $C_2$ , and  $C_3$ .  $C_1$  has a sub-parent class  $C_2$  that has two leafnodes  $N_1$  and  $N_2$  and a sub-parent class  $C_2$ , and  $C_2$  has three leafnodes  $N_3$ ,  $N_4$ , and  $N_5$ . To consider all the prerequisite leafnodes  $N_2$ ,  $N_3$ ,  $N_4$ ,  $N_5$  and  $N_6$  for pre-assessment under the parent class  $C_1$  as the *desired concept*, the logic based axioms for classification are stated as follows:

$$\forall \text{desiredConcept}(C_1) \forall N_2 \forall N_3 \forall N_4 \forall N_5 \forall N_6$$

$$[$$

$$: \exists \text{desiredConcept}(C_1) \wedge \exists \text{passed}(N_2) \wedge \exists \text{passed}(N_3) \wedge \exists \text{passed}(N_4) \wedge \exists \text{passed}(N_5) \wedge \exists \text{passed}(N_6) \Rightarrow \text{desiredConcept}(C_1). \{N_1\}. \quad (1)$$

$$: \exists \text{desiredConcept}(C_1) \wedge \exists \text{passed}(N_2) \wedge \exists \text{passed}(N_3) \wedge \exists \text{passed}(N_4) \wedge \exists \text{passed}(N_5) \wedge \exists \text{failed}(N_6) \Rightarrow \text{failed}(N_6). \quad (2)$$

$$: \exists \text{desiredConcept}(C_1) \wedge \exists \text{passed}(N_2) \wedge \exists \text{passed}(N_3) \wedge \exists \text{passed}(N_4) \wedge \exists \text{failed}(N_5) \wedge \exists \text{passed}(N_6) \Rightarrow \text{failed}(N_5). \quad (3)$$

$$: \exists \text{desiredConcept}(C_1) \wedge \exists \text{passed}(N_2) \wedge \exists \text{passed}(N_3) \wedge \exists \text{failed}(N_4) \wedge \exists \text{passed}(N_5) \wedge \exists \text{passed}(N_6) \Rightarrow \text{failed}(N_4). \quad (4)$$

$$: \exists \text{desiredConcept}(C_1) \wedge \exists \text{passed}(N_2) \wedge \exists \text{failed}(N_3) \wedge \exists \text{passed}(N_4) \wedge \exists \text{passed}(N_5) \wedge \exists \text{passed}(N_6) \Rightarrow \text{failed}(N_3). \quad (5)$$

$$: \exists \text{desiredConcept}(C_1) \wedge \exists \text{failed}(N_2) \wedge \exists \text{passed}(N_3) \wedge \exists \text{passed}(N_4) \wedge \exists \text{passed}(N_5) \wedge \exists \text{passed}(N_6) \Rightarrow \text{failed}(N_2). \quad (6)$$





$$\begin{aligned} & \forall \text{desiredConcept}(C_1) \forall N_2 \forall N_3 \forall N_4 \forall N_5 \forall N_6 \\ & : \exists \text{desiredConcept}(C_1) \wedge \exists \text{failed}(N_2) \wedge \exists \text{failed}(N_3) \wedge \exists \text{failed}(N_4) \wedge \exists \text{failed}(N_5) \wedge \exists \text{failed}(N_6) \\ & \Rightarrow \text{failed}((N_2) \wedge (N_3) \wedge (N_4) \wedge (N_5) \wedge (N_6)) \end{aligned}$$

which states *for all*  $\forall$  *desired concept* that is  $C_1$  and for the leafnodes  $N_2, N_3, N_4, N_5,$  and  $N_6$ , such that, there exists  $\exists$  in the agent beliefs the *desired concept*  $C_1$  and there exists a *failed* pre-assessment of the leafnodes  $N_2, N_3, N_4, N_5,$  and  $N_6$ , then the conclusion and recommendation for learning shall be the leafnodes  $N_2, N_3, N_4, N_5$  and  $N_6$  underneath the *desired concept*  $C_1$  submitted by the student. This type of *pre-assessment of by multiple prerequisite classes* that would involve a large number node for a subject like SQL that is reported in literature to be difficult may not be supported by *Chunking* (Casteel, 1988; Anderson, 2008): a theory that helps student to succeed. While the strategy of *pre-assessment by immediate prerequisite class* supports *Chunking*, it also allows students to complete knowledge diagnosis and get results quickly. Skills status or classification of the student is dependent on the number of prerequisite  $C_i$  classes and leafnodes  $N_{i,j}$  in a given pre-assessment. Thus, at the completion of pre-assessment by *Chunking* and having learned the materials as well, a student can choose another *desired concept* for self-testing.

For a large size of knowledge graph or ontology, the following then summarises the general form of the underlying reasoning in the pre-assessment process. Given that  $D$  is the *desired concept* that subsumes some prerequisites  $C_i$  which further subsumes some leafnodes  $N_{i,j}$  i.e.  $N_{i,j} \sqsubseteq C_i \sqsubseteq D$ ; we then state that

$$\begin{aligned} & \forall D \forall C_i \forall N_{i,j} \text{hasPrerequisite}(D, C_i) \wedge \text{hasKB}(C_i, N_{i,j}) \\ & [ \\ & : \exists D \wedge \forall \text{passed}(N_{i,j}) \Rightarrow D.\{N_D\} \\ \text{else} \\ & : \exists D \wedge \exists \text{failed}(N_{i,j}) \Rightarrow \text{failed}(N_{i,j}) \\ & ] \end{aligned}$$

where  $N_D$  represents the set of immediate leafnode instances of the *desired concept* as specified in, for example, Rule (1) from Figures 4.21 and 4.22, respectively. Note that the *desired concept*  $D \equiv C$ . This is defined in Chapter 5 using a DL language.

Again, the devised rules estimation formula comes handy in estimating the required number of classification rules. But since the ontology is non-regular, the prerequisite class nodes  $C_i$  takes a unit value, which is 1; and  $N_2, N_3, N_4, N_5$  and  $N_6$  has the total size of prerequisite leafnodes  $N = 5$  underneath the *desired concept*. Thus the number of classification  $R$  can be estimated as

$$R = C_i T^{N_{ij}} + 1$$

$$R = 1 * 2^{**5} + 1$$

$$R = 1 * 32 + 1$$

$$R = 32 + 1$$

$$R = 33$$

where 1 represents the default rule that corresponds to  $A$  in the Pre-assessment Mechanism that has no prerequisite. The leafnodes  $N_{i,j}$  are the modules in which students are tested on. On that premise, they are the nodes that counts when estimating and formulating the required number of rules depending on the given  $C_i$ . To implement the derived classification axioms above, each logical axiom has a corresponding plan in the agent program in the MAS.

As encountered during the course of this work, mapping the boolean [P, F] predicates to every leafnode  $N$  and generating the classified rules can be cumbersome. For a small number of leafnodes  $N \leq 3$ , the rules can be generated easily by hand. But for leafnodes  $N \geq 4$ , an algorithm had to be developed (*Chapter 7, Section 7.7.1*) for a program to generate the rules. The use of a program (e.g. Python) for rule generation is to ensure completeness or correctness for the rules that are deterministic: that is, exactly one rule for each episode of action or pre-assessment on the number of leafnodes  $N$ .

Each logical axiom (above) practically corresponds to one agent plan at implementation. While the rules are produced from the program written for the algorithm, the logical axioms or rules satisfy the ontological structures that are



associated. In addition, our model equation estimates the number of expected rules, for example,  $8 + 1$ ,  $16 + 1$ , or  $32 + 1$  number of rules. The model/math equation also support rule checking and ensures no case (rule) of classification is missing. In the derived logical axioms, no two axioms or rules are same. This correctness is certain via the program of parameter combination from the algorithm: the algorithm returns the expected outputs in finite steps.

### 4.9.2 Estimating The Number of Rules by Prerequisites $C_{i,j}$ and Leafnodes $N_{j,k}$ Notation in a Tree

The Figure 4.23 is a multi-dimensional knowledge graph that extends the graphs earlier presented in Figures 4.21 and 4.22, respectively. The structure presents a graph of several nodes in the horizontal plain and inter-connected nodes in the vertical traversal. All nodes are connected by a root or parent node  $C_1$ . This is to illustrate the required number of rules process. To estimate the needed number of rules, let the root node  $C_1$  be the *desired concept* (at Level 1 where a student wants to be), and its *prerequisite concepts* as  $C_2, C_3, C_4, C_5,$  and  $C_6$  (the non-terminal nodes).

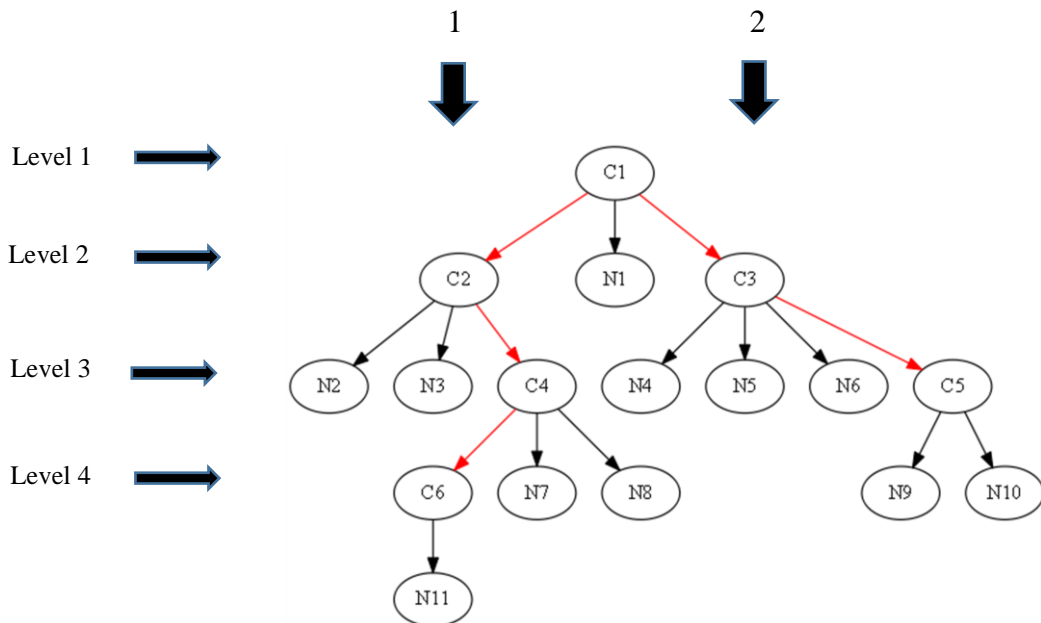


Fig.4. 23: A knowledge graph of multiple horizontal and vertical traversal

Below is the computation process of the number of classification rules for the prerequisites  $C_{i,j} N_{j,k}$ . As a *non-regular ontology*, we shall apply our model equation

$$R = 1 + \sum C_{i,j} T^{N_{j,k}}$$

Firstly, we isolate the nodes before summation:

▪ **Number of Rules Estimation Via Horizontal Navigation**

A) *Node isolation at Level 2, prerequisite class  $C_2$  to  $C_3$ , horizontal navigation through leafnodes  $N_2, N_3, N_5, N_5$  and  $N_6$ :*

$$1 + \sum [[C_{2,1}T^{N_{1,1}}, C_{2,1}T^{N_{1,2}}], [C_{2,2}T^{N_{2,1}}, C_{2,2}T^{N_{2,2}}, C_{2,2}T^{N_{2,3}}],$$

B) *Node isolation at Level 3, prerequisite  $C_4$  to  $C_5$ , horizontal navigation through leafnodes  $N_7, N_8, N_9$ , and  $N_{10}$ :*

$$[C_{3,1}T^{N_{1,1}}, C_{3,1}T^{N_{1,2}}], [C_{3,2}T^{N_{2,1}}, C_{3,2}T^{N_{2,2}}],$$

C) *Node isolation at Level 4, horizontal navigation through leafnode  $N_{11}$ :*

$$[C_{4,1}T^{N_{1,1}}]$$

*The Computation at the isolated Levels 2, 3 and 4, Horizontal navigation:*

$$R = 1 + \sum [[C_{2,1}T^2], [C_{2,2}T^3], [C_{3,1}T^2] + [C_{3,1}T^2], [C_{4,1}T^1]]$$

$$R = 1 + [1 * 2^2 + 1 * 2^3 + 1 * 2^2 + 1 * 2^2 + 1 * 2^1]$$

$$R = 1 + 4 + 8 + 4 + 4 + 2$$

$$R = 23$$

This is an estimation of the number of rules R for *pre-assessment by immediate prerequisite class* in horizontal traversal of nodes.

▪ **Number of Rules Estimation Via Vertical Navigation**

A) *Node isolation along prerequisites  $C_2$  through  $C_4$  to  $C_6$  vertical navigation to leafnodes  $N_2, N_3, N_7, N_8$  and  $N_{11}$ :*

$$R = 1 + \sum [[C_{2,1}T^{N_{1,1}}, C_{2,1}T^{N_{1,2}}], [C_{3,1}T^{N_{1,1}}, C_{3,1}T^{N_{1,2}}], [C_{4,1}T^{N_{1,1}}],$$

B) Node isolation along prerequisites  $C_3$  to  $C_5$  vertical navigation to leafnodes  $N_4, N_5, N_6, N_9$  and  $N_{10}$ :

$$[C_{2,2}T^{N_{2,1}}, C_{2,2}T^{N_{2,2}}, C_{2,2}T^{N_{2,3}}], [C_{3,2}T^{N_{2,1}}, C_{3,2}T^{N_{2,2}}]$$

*Computation along the vertical traversals:*

$$R = 1 + \sum [[C_{2,1}T^2], [C_{3,1}T^2], [C_{4,1}T^1], [C_{2,2}T^3], [C_{3,2}T^2]]$$

$$R = 1 + [1 * 2^2] + [1 * 2^2] + [1 * 2^1] + [1 * 2^3] + [1 * 2^2]$$

$$R = 1 + 4 + 4 + 2 + 8 + 4$$

$$R = 23$$

This illustrates the estimated number of rules for *pre-assessment by immediate prerequisite class* in a vertical traversal of nodes as shown with the horizontal traversal.

▪ **Number of Rules Estimation for Multiple Prerequisite Classes**

Now, let's consider the computation of the required number of rules  $R$  for the entire prerequisite classes underneath the *desired concept*  $C_1$  (Fig. 4.23). Either by vertical or horizontal traversal of the nodes as shown above, the result will be the same. From the formula  $R$ ,

$$R = 1 + \sum C_{i,j} T^{N_{j,k}}$$

and individual node isolation, and summation:

$$R = 1 + \sum [C_{2,1}T^{N_{1,1}}, C_{2,1}T^{N_{1,2}}, C_{3,1}T^{N_{1,1}}, C_{3,1}T^{N_{1,2}}, C_{4,1}T^{N_{1,1}}, C_{2,2}T^{N_{2,1}}, \\ C_{2,2}T^{N_{2,2}}, C_{2,2}T^{N_{2,3}}, C_{3,2}T^{N_{2,1}}, C_{3,2}T^{N_{2,2}}]$$

$$R = 1 + \sum [C_{2,1}T^2, C_{3,1}T^2, C_{4,1}T^1, C_{2,2}T^3, C_{3,2}T^2]$$

$$R = 1 + [1 * T^2 + 1 * T^2 + 1 * T^1 + 1 * T^3 + 1 * T^2]$$

$$R = 1 + 2^{10}$$

$$R = 1025$$

Thus, for a total of 10 leafnodes that may be considered under a *desired concept D*, 1025 is the number of classification rules that will be needed to be trained from the *passed* and *failed* boolean predicates mapping with the 10 leafnodes. Note that the value of *C* for all calculation for *non-regular ontologies* in this work equals 1.

## **4.10 Summary of Chapter**

This chapter has presented the agent based Pre-assessment System as modelled with the Prometheus methodology using the Prometheus Design Tool (PDT): a graphical agent UML for specifying agent designs from scenario development, to goal specification and refinement, to percept, message, data coupling, action, plans and their interactions. The chapter presented a student model with parameters that can obtain attributes from the student environment and then described a mechanism of pre-assessment which is the underlying strategy for diagnosing learning gap, classifying and making recommendation for students after their pre-assessments. While Gamalel-Din (2002) applied learning-by-experience, this thesis uses a classification technique via some classification rules. This is defined with first-order logic (FOL) as the reasoning process about the decision messages reached over students' skill tests. The analysis has been shown in this chapter with ontology tree models and FOL formulas. The FOL based rules are a conjunction of the <P> and <F> boolean parameter combinations mapped to leafnodes N. To support students for effective learning, *Chunking* was identified as a good educational strategy for pre-assessments and supported learning of SQL. The chapter then illustrated how our modelled equations does estimates the number of classification rules. While the *Initialisation equation* estimates the number of *classification* rules for 1) batches of immediate prerequisite class pre-assessment and 2) multiple class pre-assessment; the *polynomial equation* has been used to estimate the number of *classification* rules for batches of multiple prerequisite class pre-assessment as illustrated. In Chapter 5, the implementation of the Pre-assessment System in Jason agent language shall be presented. The chapter shall cover the real-time SQL domain ontology development with description logic, ontology construction and visualisation; and its first-order representation for agents.

# Chapter 5

## A SQL Ontology and The Pre-assessment System

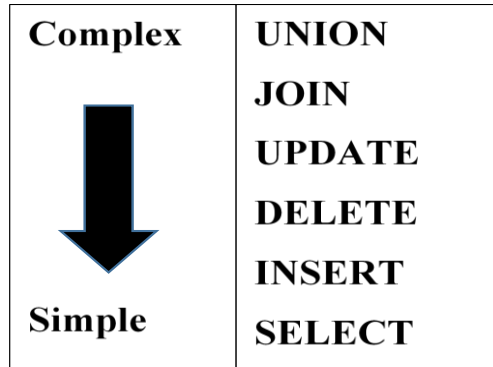
### 5. Introduction

In Chapter 4, an AOSE graphical editing tool, the PDT which is an agent UML that supports the *Prometheus* methodology was presented as employed in the specification and design of the Pre-assessment System. The chapter described the *Pre-assessment Mechanism* as a process for identifying gaps in student learning, and explained the parameters of the *Student Model* of this research and their use as predicates for: inter-agent messages, classification reasoning about students' knowledge status and first-order logic (FOL) formulas. This chapter presents the implementation of the agents of the Pre-assessment System as specified in Chapter 4 for the pre-assessment of students and inter-agent communication in the pre-assessment process. Firstly, the chapter presents an SQL learning structure, then the SQL domain ontology definition in a TBox using description logic (DL) syntax, and the different ontology models generated from the TBox. It looks at concepts relationships in Jena API ontology model and the Protégé ontology editor, then knowledge representation in FOL from the ABox assertions for agents' beliefs. The chapter also describes CArtAgO as the environment artifact for percepts observation.

### 5.1 Contextual Learning Structure

The domain context of this system is Structured Query Language (SQL) which is presented in a structured hierarchy in Figure 5.1. In a teaching-learning environment, modules are taught in an order of sequence from simple to complex as specified in a given curriculum. In a top-down approach, this is presented in the hierarchy of

complex to simple concept, namely: *UNION*, *JOIN*, *UPDATE*, *DELETE*, *INSERT*, and *SELECT* where *UNION* is the complex concept and *SELECT* is the lowest.



**Fig.5. 1:** Hierarchy of six SQL Modules Learning Structure (extended version of Ehimwenma, Beer & Crowther 2014b).

In this arrangement, a lower module is taught and learned before a higher one. Thus, any immediate-lower concept is a prerequisite to its next higher concept. The topics in this structure are the modules in which students would be pre-assessed on the Pre-assessment System to identify gaps in their learning so as to make recommendation for learning materials to assist them in closing the gaps. Thus, the Figure 5.1 presents a:

- Hierarchy in which students are pre-assessed in structured sequence. This is because in such an arrangement, one topic is taught before the next in a bottom-up approach;
- Domain for formalising a definition of ontology in SQL using a DL TBox;
- Domain in which instances of classes (topics) will be named as ABox assertions in FOL to represent knowledge structures for agents and inter-agent communication.

## 5.2 Description Logic for SQL Ontology

Description logic (DL) is a family of knowledge representation (KR). KR is the set of acquired experiences or background structure of knowledge that an intelligent system is given to function: to reason, to query, to make judgement or prediction. This sort of KR in artificial intelligence (AI) as ascertained in Baader *et al.* (2003) is usually on

methods for providing high-level description of the domain of interest or world in FOL formalism for building intelligent applications.

In the following section, a formal definition of a SQL ontology is presented using a DL syntax. The DL ontology describes the relationships between classes, classes and individuals and the constraints or restrictions on individuals. KR based on DL consists of two components: TBox and ABox (Obitko, 2007). The TBox describes terminology for the SQL ontology and the ABox introduces the individuals and their relations for representation in the Pre-assessment System.

### 5.2.1 TBox Description for a SQL Ontology

The Figure 5.2 is a TBox terminology (*hierarchical*) (Nardi & Brachman, 2003) description of concept names for a SQL domain ontology. The concept names are the *named symbols* on the left hand side of the *equivalence*  $\equiv$  symbol and are defined on the right hand side as *base symbols* (Baader & Nutt, 2003) as explained in Chapter 2. Given the DL syntax  $\exists r.C$  that a thing has a role or relation with the concept  $C$  e.g.  $\exists \text{hasChild.Lawyer}$ , and  $\exists r.\{x\}$  that a thing has some relation with a some instances e.g.  $\exists \text{citizenOf.\{USA\}}$  (Baader, horrocks & Sattler, 2003); then from the Figure 5.2, the axiom

$$\text{SqlNode} \equiv \text{SqlClassNode} \sqcap \text{SqlSubClassNode}$$

$\text{SqlNode}$	$\equiv$	$\text{SqlClassNode} \sqcap \text{SqlSubClassNode}$
$\text{LeafNode}$	$\equiv$	$\text{SqlSubClassNode} \sqcap (\exists \text{hasQuiz.Quiz} \sqcap \exists \text{hasAnswer.Answer} \sqcap \exists \text{hasContent.WebUrl}) \sqcap \neg \text{SqlClassNode}$
$\text{PrerequisiteConcept}$	$\equiv$	$\text{SqlClassNode} \sqcap \geq 2 \text{ hasKB.LeanNode} \sqcap ((\exists \text{hasPrerequisite.SqlSubClassNode} \sqcap \exists \text{isPrerequisiteOf.SqlClassNode}) \sqcup (\exists \text{hasPrerequisite.SqlSubClassNode}))$
$\text{DesiredConcept}$	$\equiv$	$\text{SqlNode} \sqcap \exists \text{hasPrerequisite.PrerequisiteConcept}$
$\text{isPrerequisiteOf}$	$\equiv$	$\text{hasPrerequisite}^-$

Fig.5. 2: TBox Description of an SQL Domain.

defines a *SqlNode* as parent class nodes and subclass nodes in this SQL domain ontology. This represents the class node concept that is required to be entered by a student as a *desired\_Concept* intended to be studied upon which some pre-assessments will be conducted.

The following axiom

$$\begin{aligned} \text{LeafNode} \equiv & \text{SqlSubClassNode} \sqcap (\exists \text{hasQuiz.Quiz} \\ & \sqcap \exists \text{hasAnswer.Answer} \\ & \sqcap \exists \text{hasContent.WebUrl}) \\ & \sqcap \neg \text{SqlClassNode} \end{aligned}$$

uses *existential restriction*  $\exists$  to define the term *LeafNode* as subclass nodes that have some quizzes, answers and web URLs (universal resource locator) via their respective *hasQuiz*, *hasAnswer* and *hasContent* relations, and also with the *classical negation*  $\neg$  symbol that leafnodes are not parent class nodes per se. The terms *Quiz*, *Answer* and *WebUrl* depicts the corresponding literals to the defined terms for every leaf node that are used for pre-assessment and recommendation.

In the axiom that involves the use of a *minimum cardinality* restriction of 2

$$\begin{aligned} \text{PrerequisiteConcept} \equiv & \text{SqlClassNode} \sqcap \geq 2 \text{ hasKB.LeafNode} \\ & \sqcap ((\exists \text{hasPrerequisite.SqlSubClassNode} \\ & \sqcap \exists \text{isPrerequisiteOf.SqlClassNode}) \\ & \sqcup (\exists \text{hasPrerequisite.SqlSubClassNode})) \end{aligned}$$

the *PrerequisiteConcept* is defined as class concepts that have at least two leaf nodes and either a *hasPrerequisite* relation to a (sub)class and a *isPrerequisiteOf* inverse or a *hasPrerequisite* relation to the (sub)class concept.

Then, the axiom

$$\text{DesiredConcept} \equiv \text{SqlNode} \sqcap \exists \text{hasPrerequisite.PrerequisiteConcept}$$

defines a *DesiredConcept* as nodes that have some prerequisite node via the *hasPrerequisite* relation, and finally,

$$\text{isPrerequisiteOf} \equiv \text{hasPrerequisite}^{-}$$



which states that the *isPrerequisiteOf* relation is the inverse of *hasPrerequisite* relation.

From the DL syntax, *named symbols*, for example *DesiredConcept* is defined. Roles or relationships such as *hasPrerequisite*, *hasKB* (Ehimwenma, Beer & Crowther, 2014a), and *isPrerequisiteOf* are also defined. While the *DesiredConcept* is unary predicate for a desired concept in a FOL statement for agents' communication, the *hasPrerequisite*, *hasKB*, and *isPrerequisiteOf* are binary predicates between classes and individuals.

### 5.2.2 SQL Individuals in Description Language

Individuals values, as ascertained in Baadar & Nutts (2003) are not only meant to be asserted in ABox. They can be instantiated also in a TBox. By implication, the DL SQL ontology defined above can have instances of individuals defined within it, for example, the *DesiredConcept* term can also be instantiated as:

$$\text{DesiredConcept} = \{\text{insert}\} \sqcap \text{hasPrerequisite}.\{\text{select}\} \\ \sqcap (\text{hasKB}.\{\text{selectWhere}\} \sqcap \text{hasContent}.\{\text{http://...}\})$$

which states, insert is a desired concept that has a hasprerequisite relation with select that has a knowledge base with the hasKB relation with selectWhere that has a URL link with the hasUrl relation.

### 5.2.3 ABox Assertion for a SQL Ontology

ABox contains assertion knowledge called *ground fact* which are individuals and their properties (Rudolph, 2011). Based on the SQL learning structure (Fig. 5.1), the class instances of the *desired\_Concepts* can be declared as:

```
DesiredConcept = {union, join, update, delete, insert, select}
```

and the set of leaf node instances which are:

```
LeafNode = {unionAll, unionDistinct, selfJoin, fullJoin,
innerJoin, UpdateSelect, updateWhere, deleteSelect,
deleteWhere, insertSelect, insertWhere, selectWhere,
selectAll,selectOrderBy, selectDistinct}
```

Similar to the examples shown in literature as in C (a) that a belongs to the interpretation of C e.g. *father(peter)*, and R (b, c) that c is a filler for the role R for b (Baadar & Nutts, 2003), the following ABox assertions are then stated, in their unary and binary predicate e.g.

*desiredConcept(update)*

that Update is a *desired\_Concept*; and that

*hasPrerequisite(update, delete)*

Update has prerequisite Delete, an inverse relation

*isPrerequisiteOf(delete, update)*

which states Delete is a prerequisite of Update; and another *hasKB* connected predicate relation

*hasKB(update, updateSelect)*

that Update has KB UpdateSelect

are *ground* (first-order) atomic formula for Jason agent language beliefs representation. Such set of beliefs are the agent's knowhow of its world (Bordini, Hubner & Tralamazza, 2006).

### 5.3 Digraph analysis of the Description Logic SQL Ontology Model

Based on the SQL TBox description, different ontology models were created to visualise the knowledge modules in the domain of SQL and the modules relationships to each other. Using graphical analysis, the models that are created from ABox assertion are given below as: *regular* ontology and *non-regular* ontologies (section 5.3.1 and 5.3.2). The ontology models are directed graphs where the directed links between nodes indicates navigation. The graphs contain six class node concepts according to the SQL learning structure in Figure 5.1, with the *hasPrerequisite* relation between class nodes, and *hasKB* relation between a class and its leaf nodes.

### 5.3.1 A Regular SQL Ontology

A *regular* ontology is an ontology with an equal number of leaf-nodes across all its parent class nodes in its tree (Ehimwenma, Beer & Crowther, 2015a). The Figure 5.3 is a regular ontology of a linear configuration from top to bottom with two leaf nodes across all parent class nodes. An immediate lower node is a prerequisite to its top node.

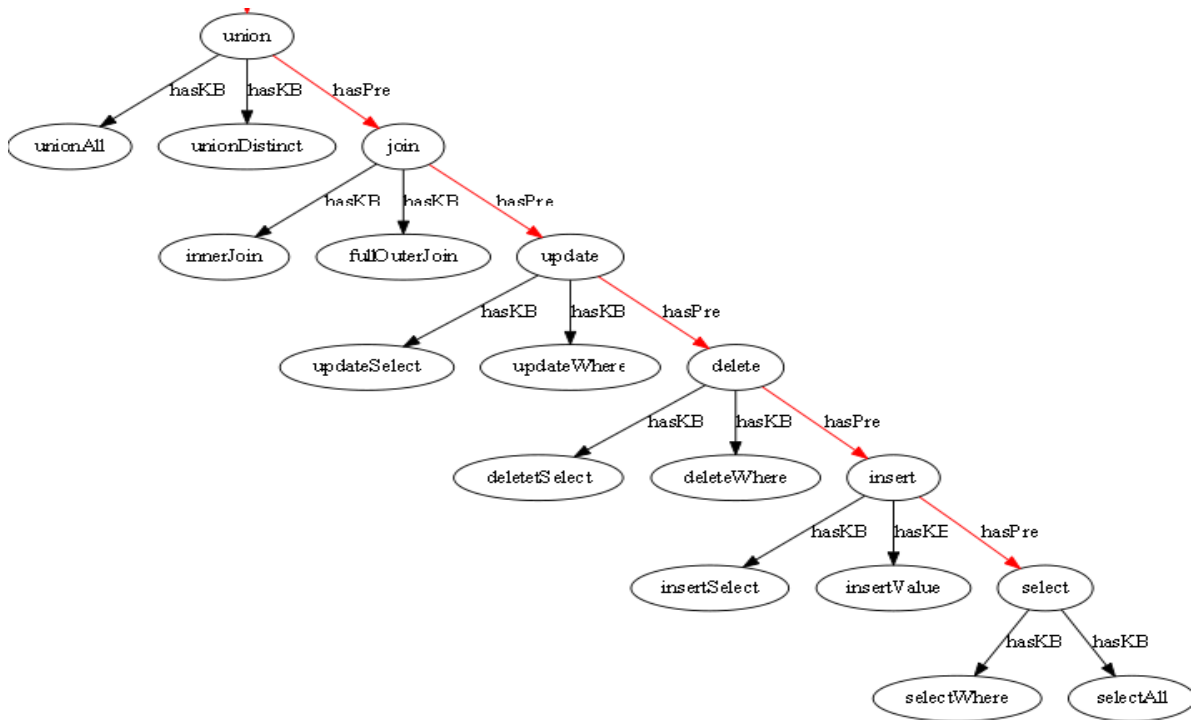


Fig.5. 3: A regular ontology of two leaf nodes per parent class node.

The relation linking two parent class nodes (top and immediate next) is the *hasPrerequisite* binary relation. The desired concepts (which are parent class nodes) has two leaf nodes with the *hasKB* relation, and other edge labelled the *hasPrerequisite* relation linking other class nodes in the hierarchy which are themselves *DesiredConcept* as defined in the DL syntax of Figure 5.2.

### 5.3.2 Non-Regular SQL Ontology Model

Recall that in the DL syntax (Fig. 5.2) a *minimum cardinality* constraint of at least two leaf nodes per parent class node was defined. A varying amount of leaf nodes across parent class nodes in an ontology constitutes a *non-regular* ontology. In the Figure 5.4,

the ontology has a parent class node that has more number of leaf nodes than other parent nodes in the ontology.

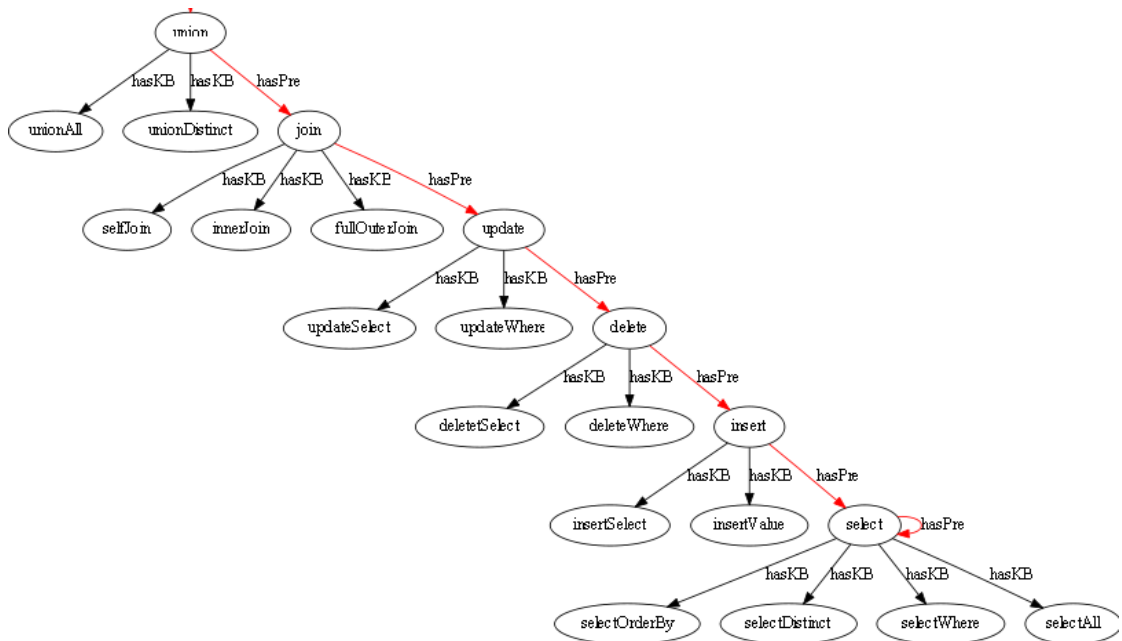


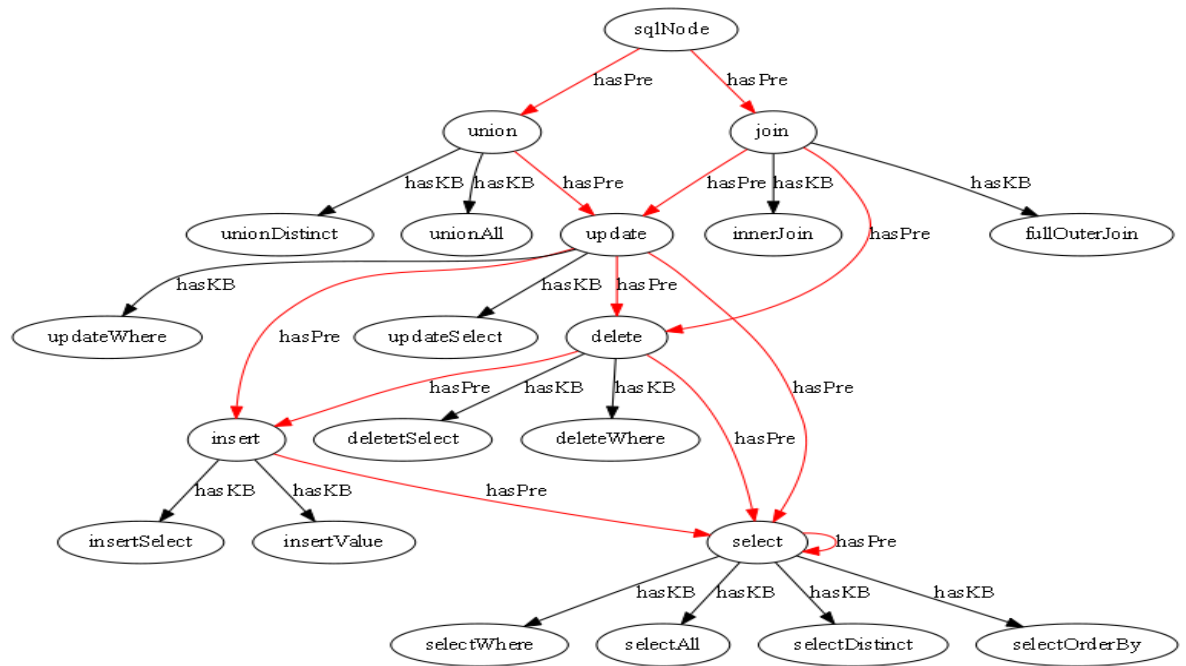
Fig.5. 4: Linear ontological model from the TBox. SELECT is reflexive.

While other parent nodes have two leaf nodes, the select concept has four leaf nodes. This is a valid representation as specified by the description in the TBox given the minimum cardinality of leafnodes  $N \geq 2$ .

Unlike the Figures 5.3 and 5.4 that has a single relation between a desired class concept and its prerequisite class, in Figure 5.5 is a model with, for example, two *hasPrerequisite* directed relations from a parent class to other parent classes. This model places two parent classes at the level e.g. Union and Join. But in teaching and learning, one unit of lesson must be taught before another. In that case, the Figure 5.5 model does not validate the ordered sequence of the concepts provided in Figure 5.1, but the model however satisfies the TBox definition in Figure 5.2. Which is also true of the Figures 5.3 and 5.4 including Figure 5.5 that satisfies the axiom

$$\equiv \text{SqlClassNode} \sqcap \geq 2 \text{ hasKB. LeafNode} \\ \sqcap (\exists \text{hasPrerequisite. SqlSubClassNode})$$

As a type of formative assessment system that enables students to make a choice of their desired learning concept, pre-assessment exercises that determines whether a student should learn his or her desired concept or not must be in ordered sequence. This is to avoid any gaps in the hierarchy of learning structure.



**Fig.5. 5:** A non-linear hierarchy of the SQL learning structure. But some parent class nodes are not connected in sequence according to Fig. 5.1.

Another model of the TBox is that which is presented in Figure 5.6, a model where two different property relations: *hasPrerequisite* and *isPrerequisiteOf* are used as connected links between class nodes. While the *hasPrerequisite* shows the navigation from a top level concept of learning to a lower-level concept, the *isPrerequisiteOf* relation presents the connectedness from a lower-level knowledge concept to a top level concept.

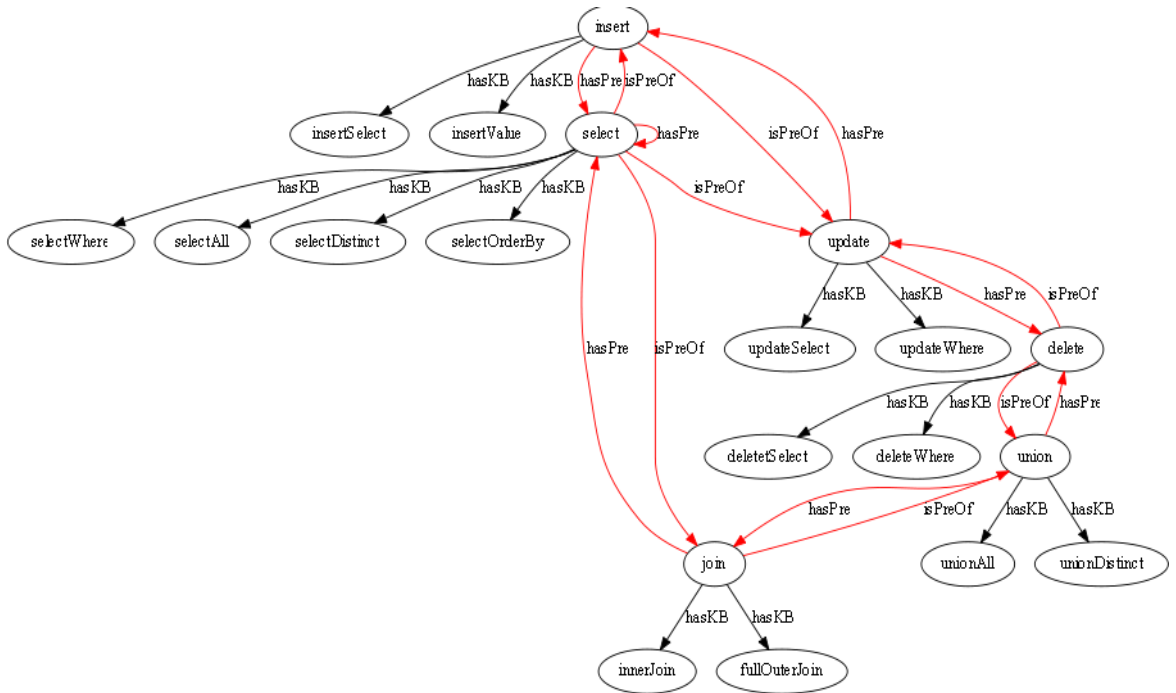


Fig.5. 6: A variant ontology model of the TBox description and its navigation. But not in the structured sequence presented in Fig. 5.1

The *isPrerequisiteOf* is the inverse property or relation to the *hasPrerequisite* property.

The Figure 5.6 satisfies the axiom

$$\equiv \text{SqlClassNode} \sqcap \geq 2 \text{ hasKB. LeafNode} \\ \sqcap ((\exists \text{hasPrerequisite.SqlSubClassNode} \sqcap \exists \text{isPrerequisiteOf.SqlClassNode})$$

option of the definition of the PrerequisiteConcept in the TBox, such that any class node that has a *hasPrerequisite* must have a *isPrerequisiteOf* relation. The drawback of the Figure 5.6 ontology model is the infinite loop traversal across parent class nodes such that the knowledge engineer will need to determine a start point and an end point that are connected for pre-assessment.

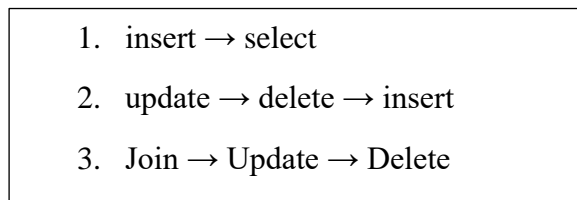
### 5.4 Navigation of Ontology Nodes

In a standard curriculum, teaching and learning is sequential and ordered, simple to complex, from one concept to another, *see Figure 5.1*. The various graphical ontology models visualised so far from the TBox has shown how a DL definition is used to describe a body of knowledge and the relationships between concepts. Roles or binary relations specified connection between nodes. In directed graphs, these relations provide a sense of navigation from node to node. For instance, the binary property

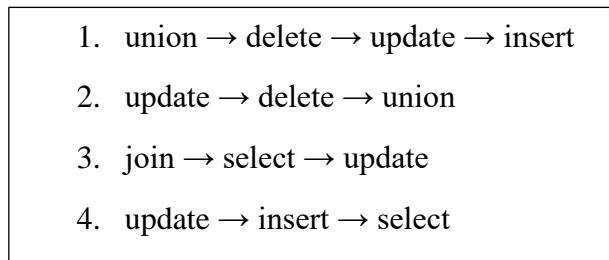
relations (e.g. Fig. 5.1, 5.2), showed possible navigation path through which concepts are linked for pre-assessment. This can be established either on the strategy of:

- *Pre-Assessment By Immediate Prerequisite Class*; or
- *Pre-Assessment By Multiple Prerequisite Classes*;

as described in Chapter 4. The directed links in the ontology models are the navigation paths from one class node concept. In Pre-assessment System of this study, the binary property depicts the manner in which agent !achievement goals are programmed to carry out the pre-assessment of students' SQL knowledge. For example, the Figure 5.7 shows the *hasPrerequisite* relation navigation based on Figure 5.2, and Figure 5.8 navigation that comprise the *hasPrerequisite* and *isPrerequisiteOf* relations based on Figure 5.6.



**Fig.5. 7:** Illustrating navigation strategy for agent !achievement goal.



**Fig.5. 8:** Illustrating navigation strategy based on directed links between class nodes. Yet contrasts the structured sequence in Fig. 5.1.

While the Figure 5.6 reflects a model of the TBox definition, it does not reflect the sequence of the SQL learning structure in Figure 5.1; e.g.

update → delete → union

which implies that: with *update* as *desired\_Concept*, pre-assessment is on the *delete* and the *union* concepts. In ABox assertion for ontologies and pre-assessment, it should

follow the order of the specified curriculum, like the navigation of the Figure 5.7. But not with the gap of a missing concept as in

update → insert → select

where the delete concept is not connected in that order. While item 1, in the Figure 5.7, is of the *Pre\_Assessment By Immediate Prerequisite Class* strategy, others are of the *Pre\_Assessment By Multiple Prerequisite Classes* as outlined in Chapter 4.

Every parent class node has its leaf nodes. The *insert* concept for instance, has its leaf node concepts named as: *insertValue* and *insertSelect*. These are the unit of lessons in which SQL skills are tested to ascertain whether there is a gap in learning before proceeding to the *insert* concept. As defined in the TBox,

$$\begin{aligned} \text{LeafNode} \equiv & \exists \text{hasQuiz.Quiz} \\ & \sqcap \exists \text{hasAnswer.Answer} \\ & \sqcap \exists \text{hasContent.WebUrl} \\ & \sqcap (\neg (\text{SqlClassNode} \sqcup \text{SqlSubClassNode})) \end{aligned}$$

all leaf nodes have their respective literals, which are the *quizzes*, *answers* and *url* data that are specified with the: *hasQuiz*, *hasAnswer* and *hasContent* relations, respectively. The *LeafNode* axiom is then explicitly expanded in Figure 5.9. The literals (*quiz*, *answer* and *url*) in rectangular shapes are *String* data values that are used for the pre-assessment, release of learning materials, and for inter-agent communication in the MAS.

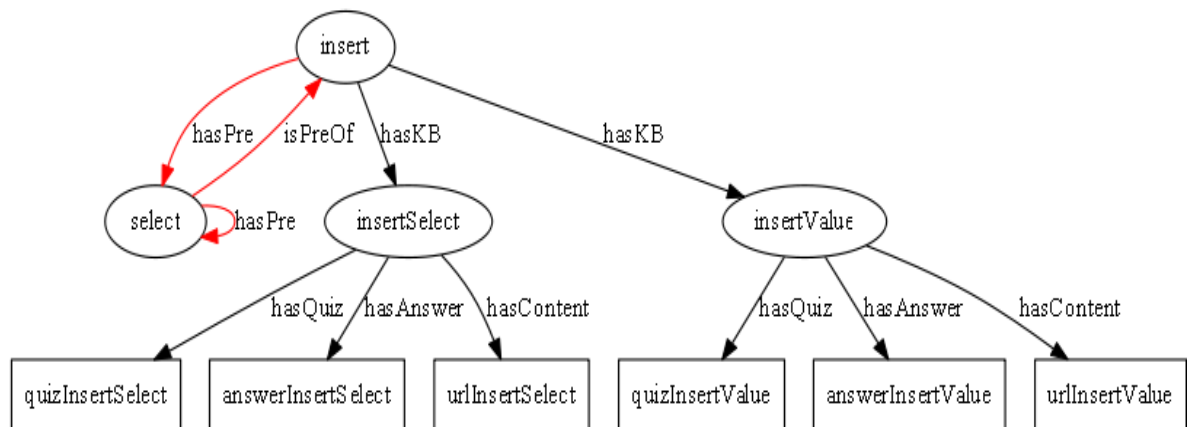


Fig.5. 9: The insert class example with its leaf node and literal (or data) nodes.



The *quiz* and *answer* literals are beliefs initialised in the BB of the agent *agSupport*: the agent that pre-assesses students, take decisions on their answer responses to quizzes, and communicates the *pass* or *fail* predicate decision statement to the agent *agModelling* (the *classifier*) for classification. The classification process which is the categorisation of student learning and recommendation of appropriate learning material(s) was represented in first order logic (FOL) formulas as the process of reasoning by the *classifier* agent in Chapter 4.

## 5.5 Ontology Building Tools: Jena API and Protégé

### Ontology Editor

An ontology is a description of things and their relationships (Gruber 1993; 1995). Ontology is a way of organising and representing knowledge. The preceding sections of this chapter has defined, and analysed a SQL learning structure. This section thus presents the use of Jena ontology API and the Protégé ontology editor in building ontologies. After the ontology construction, the OWL (web ontology language) ontology is parsed in Jena RDF API to show the compatibility of OWL and RDF KR. It is pertinent to state that the purpose is not to query ontology repository such as Protégé or Jena ontology models, but to amongst other objectives depict the *subject*, *predicate*, *object* format for FOL representation.

#### 5.5.1 Constructing ontologies in Jena API

RDF is a graph database. RDF defines resources as connected graphs in their *subject*, *predicate*, *object* form. A class (*subject* or *object*) and relation (i.e. *predicate*) are all resources in RDF.

From the ontology models (i.e. Figure 5.3, 5.4 or 5.5), let us consider a cross-section of class concepts that comprises *Delete*, *Insert* and *Select* and their relations to illustrate an RDF ontology model. Using TURTLE as the output syntax in Jena (Fig. 5.10), the output shows that *delete* has a CLASS relation with *Insert*, and a ROLE property or relation with *deleteWhere* and *deleteSelect*. Then *Insert* that also have a CLASS relation with *Select*, and a ROLE relation with *insertWhere* and *insertSelect*.

RDF data structure does not support unary predicate relation. But a set of triple that is expressed as logical formulas  $p(a, b)$  (see Chapter 2).

```

<delete> <http://www.w3.org/2001/vcard-rdf/3.0#CLASS> <insert> ;
        <http://www.w3.org/2001/vcard-rdf/3.0#ROLE>
        "deleteWhere", "deleteSelect" .

<insert> <http://www.w3.org/2001/vcard-rdf/3.0#CLASS> <select> ;
        <http://www.w3.org/2001/vcard-rdf/3.0#ROLE>
        "insertWhere", "insertSelect" .

<select> <http://www.w3.org/2001/vcard-rdf/3.0#ROLE>
        "selectOrderBy", "selectDistinct", "selectAll", selectWhere".
    
```

Fig.5. 10: Jena ontology rendered in Turtle syntax.

### 5.5.2 Protégé Ontology Tool

Like Jena, Protégé ontology editor constructs and renders ontology in different output syntax. An example is the RDF/XML syntax. Using the same cross-section of class concepts that comprise the *Delete*, *Insert* and *Select*; Protégé, an OWL tool is used to visualise the classes and their relations (Fig. 5.11).

In furtherance, to establish the backward compatibility of OWL syntax to RDF, the OWL ontology rendered in RDF/XML format is parsed in Jena using the Turtle format.

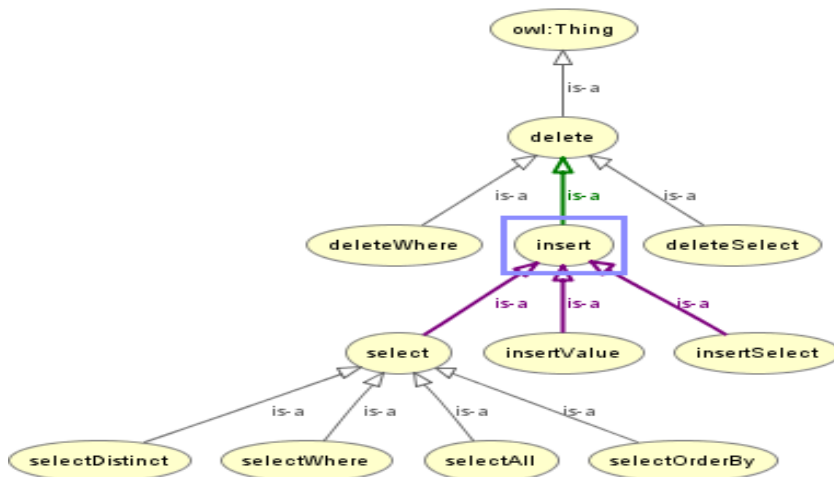


Fig.5. 11: A cross-section of the concepts: DELETE, INSERT and SELECT in structured of Figure 5.1.

Ontologies rendered in RDF/XML or OWL/XML are in their fully qualified URI (universal resource identifier). But in parsing the OWL file in Jena, TURTLE syntax also output the ontology only in their given resource names, with additional information such as the *owl:class*, and an *rdfs:subClassOf* relation (Fig.5.12).

---

```
<http://www.semanticweb.org/lette/ontologies/sql/delete>
  a      owl:Ontology .
:delete a      owl:Class .
:deleteSelect a      owl:Class ;
      rdfs:subClassOf :delete .
:deleteWhere a      owl:Class ;
      rdfs:subClassOf :delete .
:insert a      owl:Class ;
      rdfs:subClassOf :delete .
:insertSelect a      owl:Class ;
      rdfs:subClassOf :insert .
:insertValue a      owl:Class ;
      rdfs:subClassOf :insert .
:select a      owl:Class ;
      rdfs:subClassOf :insert .
:selectAll a      owl:Class ;
      rdfs:subClassOf :select .
:selectWhere a      owl:Class ;
      rdfs:subClassOf :select .
:selectOrderBy a      owl:Class ;
      rdfs:subClassOf :select .
:selectDistinct a      owl:Class ;
      rdfs:subClassOf :select .
:hasKB a      owl:ObjectProperty ;
      rdfs:domain :delete , :select , :insert ;
      rdfs:range :insertSelect , :deleteSelect , :deleteWhere ,
                :selectOrderBy , :selectWhere , :selectAll ,
                :insertValue , :selectDistinct .
:hasPrerequisite a      owl:ObjectProperty ;
      rdfs:domain :select , :insert , :delete ;
      rdfs:range :select , :insert .
```

---

**Fig.5. 12:** Protégé OWL ontology using Turtle syntax from Jena API.

For instance, the statement

```
:insert a owl:Class ; rdfs:subClassOf :delete .
```

is a class to class relation that states `insert` is an owl class and by the `rdfs` property it is an subclass of `delete`. This class to class relation also applies to other class concepts in the learning structure (Fig. 5.1). Similarly, in the following statement

```
:deleteSelect a owl:Class ; rdfs:subClassOf :delete .
```

the `deleteSelect` concept is an owl class and a subclass of the `delete` concept. In the TBox (Fig.5.2) the leaf node is defined as a subclass of a class concept,

but not amongst the *PrerequisiteConcepts* that has the *hasPrerequisite* property. In the OWL ontology the relationship between classes is established with the *hasPrerequisite* property, and that of a class node to leaf node by the *hasKB* property. The *hasPrerequisite* and *hasKB* relations are ObjectProperty (Horridge et al. 2004) relations that have their respect range and domain concepts listed alongside in the illustrated TURTLE syntax (Fig. 5.12).

```

hasPrerequisite(Union, Join) [ont(sql)].
    hasKB(join, outerJoin) [ont(sql)].
    hasKB(join, innerJoin) [ont(sql)].
hasPrerequisite(Join, Update) [ont(sql)].
    hasKB(update, updateSelect) [ont(sql)].
    hasKB(update, updateWhere) [ont(sql)].
hasPrerequisite(Update, Delete) [ont(sql)].
    hasKB(delete, deleteSelect) [ont(sql)].
    hasKB(delete, deleteWhere) [ont(sql)].
hasPrerequisite(Delete, Insert) [ont(sql)].
    hasKB(insert, insertSelect) [ont(sql)].
    hasKB(insert, insertWhere) [ont(sql)].
hasPrerequisite(Insert, Select) [ont(sql)].
    hasKB(select, SelectWhere) [ont(sql)].
    hasKB(select, SelectAll) [ont(sql)].
hasPrerequisite(Select, Select) [ont(sql)].

```

Fig.5. 13: A Regular SQL ontology

Having semantically analysed different ontology models from the TBox definition and ABox assertions, the FOL representation of knowledge for the Pre-assessment System (agents) given the ABox assertion in the hierarchy of the SQL learning structure (Fig. 5.1) is stated as follows (Fig. 5.13): which is a representation for a regular ontology i.e. an ontology with equal number of leaf nodes per parent class across an ontology tree with every statement annotated with *[ont(sql)]* as SQL ontology. In the following section, the pre-assessment System is presented with its agents and CArAgo environment.

## 5.6 The Pre-assessment System

The Pre-assessment System is a multiagent system (MAS) of five component agents. The agent oriented programming (AOP) language for its implementation is *Jason*, a variant of *AgentSpeak language*. The choice is based on the analysis in Chapter 3 that *Jason AgentSpeak* is a:

- first-order logic (FOL) knowledge representation language, with beliefs in Prolog-like data structure; and
- supports speech acts based inter-agent communication using performatives or communicative acts.

*Jason* is a reactive AOP language. Thus, the Pre-assessment System is also a reactive MAS. The Pre-assessment System obtains percepts from the student (environment) with CArtaGO: the reactive interface, and communicates all percepts for the pre-assessment and classification of students' true state of learning. The agents of the Pre-assessment System as configured in Jason AgentSpeak language are shown as follows in Figure 5.14:

- *Agent agInterface*: The agent that creates the CArtaGO artifact and observes it.
- *Agent agSupport*: The agent that pre-assesses students' knowledge and make either a *pass* or a *fail* decision.
- *Agent agModelling*: The agent that classifies students' knowledge by matching its classification rules to the *pass* or *fail* decision messages received.
- *Agent agModel*: The agent that keeps persistent beliefs of all pre-assessment activities.
- *Agent agMaterial*: The agent that recommends learning materials.

As indicated in Chapter 2, these five cooperative agents are comparable to the integrated multi-part components of a recommender system e.g. El Mabrouk, Gaou & Rtili (2017); or the Padayachee (2002) Classical Four Model ITS architecture and micro-society of agents for solving a problem, respectively. The five agents and their functions were first identified and specified at the *Architectural Design* phase in Chapter 4 (e.g. Figures 4.5, 4.6, and 4.7) along with their roles, percepts, actions, messages, and plans specified at the *Detailed Design* phase in Figures 4.11 to 4.16.

```

MAS pre_assessment {
    infrastructure: Centralised
    environment: c4jason.CartagoEnvironment
    agents:
        agInterface agentArchClass c4jason.CAgentArch;
        agSupport; //pre-assessment
        agModelling; //classifier
        student beliefBaseClass jason.bb.TextPersistentBB;//agModel
        agMaterial; //ontology

    classpath: "../../../lib/cartago.jar"; "../../../lib/c4jason.jar";
}

```

Fig.5. 14: Snapshot of Agents creation and configuration in the Pre\_assessment MAS Project in Jason.

### 5.6.1 CArtaGO + Jason

Firstly, in Figure 5.14, the MAS project is declared to run on the *Centralised* infrastructure of Jason. This infrastructure as stated in Chapter 3 enables Jason agents to run on a local machine. The

*environment: c4jason.CartagoEnvironment*

is a declaration of a default workspace environment, meant for the agent *agInterface* in the following declaration:

*agInterface agentArchClass c4jason.CAgentArch*

to create the CArtaGO (Ricci, Piunti, Viroli, 2011) environment for percept observation at the start of the Pre-assessment MAS. This class is a Jason library file that can be assigned to agent(s) to construct a CArtaGO environment. Also configured are the:

- 1) *cartago.jar* and *c4jason.jar* libraries in the declared class path;
- 2) *c4jason.Environment* as the environment declaration.

These files are required for the MAS to work within the CArtaGO environment. The *Jason* infrastructure selected to run the MAS is the *Centralised* infrastructure, and the

*Student beliefBaseClass Jason.bb.TextPersistentBB*

is a text persistent belief base (BB) for the agent *agModel* (student) to permanently keep the pre-assessment activities of students. The IDE (integrated development environment) used for developing the Pre-assessment System is the *jEdit* for coding or programming agents in Jason.

## 5.7 The Pre-assessment System Environment

In Monette (2014) model of designing an interactive agent system for human learning, the system comprises four components, namely:

- *Environment* which implies a *set of students*;
- *Sensor* which is the *keyboard*;
- *Actuator* which implies the *screen display* (e.g. exercises, suggestions and corrections);
- *performance measure* that evaluates *student's score*.

Based on the Monette (2014), Figure 5.15 presents the description of the facilities in the Pre-assessment MAS environment. The environment of the Pre-assessment System is a partially observable environment (Wang, 2014). According to Wang, environments where agent are not directly situated are partially observable to the agent. In the Monette (2014) model for the design of an interactive tutor, students and school are prescribed as an agent environment. The *Sensor* facility is enabled by the CArTAgO workspace artifact for the MAS to observe events that are external to it. The observable events are text-based SQL topics i.e. desired concept of students and their SQL answer queries, where the answers (correct and incorrect SQL queries) are open-ended inputs from the keyboard. The *actuators* are the output screen in which an agent can display information to the environment, and the *performance measure* is the accurate classification of students' SQL knowledge status.

---

### Designing a Pre-assessment System:

<b>Performance Measure</b>	Making accurate classification
<b>Environment</b>	Set of students (i.e. cognition), school
<b>Sensors</b>	Keyboard (i.e. entering desired concept, answer to quizzes)
<b>Actuators</b>	Display screen, output console (i.e. quiz, URL referrals, feedback)

---

**Fig.5. 15:** Facility of the Pre-assessment System Agent (Based on Monette, 2014)

The Monette (2014) model emphasises the Russel & Norvig (2010) Structure of Simple Reflex Agent by specifying the facilities that constitutes an agent based system's *environment*, *sensors* and *actuators*.

## 5.8 Programming CArtaGO for Open-Ended Percepts

An agent can be reactive (Wooldridge & Jennings, 1995; Chin et al. 2014, *see Chapter 3*): from the context of action and reaction, agents continuously perceive inputs from their environment. In this view, agent activities are both perception and action. The Pre-assessment System is a *Vertical (one pass) Architecture* such that the percept received by an agent at the interface is communicated from agent to agent across the MAS. Each agent is programmed with individual plans to carry out some specific functions in the process of pre-assessment. From amongst its plans, an agent selects the plan whose plan *context* satisfies the incoming percept(s), and react subsequently to the actions in the body of plan.

The Pre-assessment System uses CArtaGO to observe desired concept and corresponding SQL answer queries to quizzes as *percepts* from a real-time student. Agents perceive events through sensors as collectors of environment stimuli. In CArtaGO, sensors are program structures provided in the infrastructure that agents can create, and use for directing information flow (Ricci, Viroli & Omicini; 2006). The *getObsProperty* (Ricci, Viroli & Omicini; 2006) (Fig. 5.16) in CArtaGO is the computational function in which an agent can perceive and take action that could change its belief and the beliefs of other agents. The sensors used in CArtaGO for obtaining input *percepts* are object-oriented programming methods in Java.

In this work, CArtaGO was configured and assigned to the agent *agInterface*. As a *goal*, the agent *agInterface* would create artifact and monitor its states. Given the *focus* function (Piunti, Ricci, Boissier & Hübner, 2009), agent *agInterface* is committed to the long term activity of observation of that environment (*see full listings in Appendix C.2.2*). The base artifact *class* provides basic functionalities to link GUI events to the artifact operations. Figure 5.16 shows a snapshot definition of the *String* type of *percepts* observable in the MAS.



```

...
        @OPERATION void setValue(String value){
            value = frame.getText();
            getObsProperty("value").updateValue(getValue());
        }
        private String getValue(){
            return frame.getText();
        }
...

```

**Fig.5. 16:** A Slice of the Java Code that gets Percept through human interaction in CArtAgO.

## 5.9 The Agents of the Pre-assessment System

In the following sections, a detailed description and functions of the component agent of the pre-assessment system is presented.

### 5.9.1 Agent *agInterface* and Percept Observation

In this system, the agent *agInterface* creates the GUI using the *PreassessmentGUI* class that extends the *GUIArtifact* (Fig. 5.17) and observes the dynamic user inputs. In Figure 5.18, the first *plan* with the triggering event *!create\_gui* is the agent *agInterface* *achievement* goal to create the artifact at the *start* of the MAS. The adoption of this goal results in the creation of the GUI text interface shown in Figure 5.19.

Subsequently, the second plan with the triggering event *+value(V)* is the agent sensor, and in its plan *context* is a number of selective inputs that are expected to be entered from the artifact text area. This *context* is a *pre-condition* that contains the SQL learning concepts that must be submitted or satisfied before the *body* of that plan can be executed, in this case to communicate the *percept* to the agent *agSupport*. For example, when agent *agSupport* receives a desired concept, it releases a quiz of the prerequisite concept.

On the third plan with same triggering event *+value(V)* like the second plan, the agent does not expect a null or empty input. A *String* data type must be entered for the plan to be executed as defined in the *PreassessmentGUI* class. These *Strings* are both the SQL concepts and their respective SQL queries to prerequisite assessments.

```

package c4jexamples;
import javax.swing.*;
import java.awt.event.*;
import cartago.*;
import cartago.tools.*;
/**
definition of the GUI artifact for the agent to create and observe
at run time.
*/
public class PreassessmentGUI extends GUIArtifact {
    private MyFrame frame;
    public void setup() {
        frame = new MyFrame();
        linkActionEventToOp(frame.submitButton,"submit");
        linkKeyStrokeToOp(frame.text,"ENTER", "updateText");
        linkWindowClosingEventToOp(frame, "closed");
        defineObsProperty("value", getValue());
        frame.setVisible(true);
    }
...

```

**Fig.5. 17:** Snapshot of the PreassessmentGUI CArtAgO Artifact

```

// agent agInterface
!create_gui. //goal to create GUI artifact

/* plan */

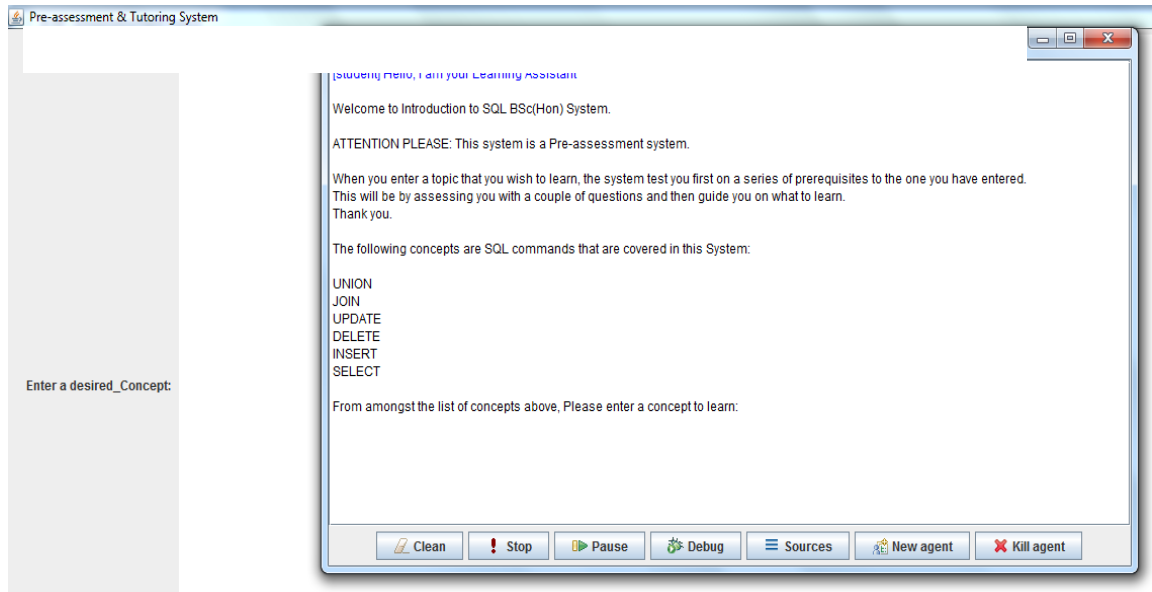
//creating GUI
+!create_gui
  <- makeArtifact("gui", "c4jexamples.PreassessmentGUI",[],Id);
  .
  .
  focus(Id). //long term focus on artifact observation

// perceiving student's desired concept from GUI
+value(V)[source(percept)] : value("SELECT") | value("INSERT") |
value("DELETE") | value("UPDATE") | value("JOIN") | value("UNION")
  <-println("The topic you have entered to learn is: ", V);
  .send(agSupport, tell, value(V));
  .println("").

// perceiving student's answer from GUI
+value(V)[source(percept)] : not value("")
  <-println("The answer you have provided is: ", V);
  .println("");
  .send(agSupport, tell, value(V));
  .wait(600000).

```

**Fig.5. 18:** A slice of Jason plans that creates observable artifact and percept communication



**Fig.5. 19:** CArtAgO artifact for Agent Percept and User Interaction. With overlapping MAS output or display console. The output console prompts the user for inputs when the MAS is started (Ehimwenma, Beer & Crowther, 2015a).

### 5.9.2 Agent *agModelling* and Classification

The agent *agModelling* is the *Classifier* agent of this system as specified with the PDT systems design in Chapter 4. Classification in the context of this work is the reasoning over the aggregate of decision messages from the agent *agSupport* after pre-assessment for the accurate and selective categorisation of students for learning materials. These messages are those predicated with the *desiredConcept* <D>, *passed* <P> or *failed* <F> parameters as prescribed in the *Student Model* (Chapter 4). For every pre-assessment quiz carried out by the agent *agSupport* (like the human teacher) on a student, the classifier agent is always updated to begin the process of reasoning over the messages based on the FOL pre-condition statements in its plan *context*. In Jason, the format for adopting the plan, classifying, and making recommendation for learning material is stated as (Ehimwenma, Beer & Crowther, 2016a):

```
+!recommend_material : set_of_profile_parameters
    <- recommended_material.
```

where *+!recommend\_material* represents the triggering message from the sender agent *agSupport* with a `tell` performative; *set\_of\_profile\_parameters*, the pre-conditions

that are matched with every updated beliefs received by a *tell* performative, and the *recommended\_material* as the message content with an *achieve* performative to the learning material agent *agMaterial* to be committed to achieving and releasing URL materials.

- **One vs. All Multiple Classification**

Classification as stated in Chapter 2 is predicting the correct class of an object or data after the data goes through a classifier(s) (Rifkin & Klautau, 2004; Marsland, 2014). In this research, each student skills data is proposed to belong to a single class depending on the student's *desired\_Concept* and number of prerequisite leafnodes *N*. One vs. All classification refers to the agent *agModelling* action of matching the rules in a plan *context* with beliefs and selecting a plan from amongst the number of plans to classify a student. That is, the agent decides a single accurate class and recommend suitable learning material. This is after a collection of decision statements of many observations (e.g. answer activities) from a sender agent. Then the student is presented what to learn at the end of the pre-assessment session. The agent *agModelling* has a number of first-order predicate (*passed* or *failed*) rules that are based on the number of leaf nodes under a desired concept.

As mentioned earlier, two pre-assessment strategies have been identified given the pre-assessment mechanism in Chapter 4: the *pre-assessment by immediate-next prerequisite class* is supported by the educational theory of *Chunking* (Casteel, 1988; Anderson, 2008) as discussed in Chapter 2. With a regular ontology structure, the pre-assessment system was implemented. On observing the DELETE desired concept, a slice of the rules or plans that classifies students are given in Figure 5.20. The literals in the predicate statements are in natural language that clearly represents a student's performance on the leaf nodes *insertSelect* and *insertValue* concepts.

The classifier agent *agModelling* has no initial beliefs. But updated beliefs that are communicated by the agent *agSupport*. From aggregated beliefs, plan *context* is matched and the plan selected. The updated beliefs are an accumulation of <D>, <P>, and <F> predicate statements in the course of a student's engagement with the MAS. They correspond (as shown in Figure 5.20) to the  $d(C_x)$ ,  $p(N_x)$  and  $f(N_x)$  predicate combinations in the FOL rules formulated in *Chapter 4, section 4.8*.

```

@d1
+!recommendMaterial[source(agSupport)] : desired_Concept("DELETE")[source(agSupport)]
    & passed("The student has passed the INSERT with SELECT question.")
    & passed("The student has passed the INSERT with VALUE question.")
    <- .send(agMaterial, achieve, hasPrerequisite(delete, insert)).

@d2
+!recommendMaterial[source(agSupport)] : desired_Concept("DELETE")[source(agSupport)]
    & passed("The student has passed the INSERT with SELECT question.")
    & failed("The student has NOT passed the INSERT with VALUE question.")
    <- .send(agMaterial, achieve, has_KB(insert, insert_value)).

@d3
+!recommendMaterial[source(agSupport)] : desired_Concept("DELETE")[source(agSupport)]
    & failed("The student has NOT passed the INSERT with SELECT question.")
    & passed("The student has passed the INSERT with VALUE question.")
    <- .send(agMaterial, achieve, has_KB(insert, insert_select)).

@d4
+!recommendMaterial[source(agSupport)] : desired_Concept("DELETE")[source(agSupport)]
    & failed("The student has NOT passed the INSERT with SELECT question.")
    & failed("The student has NOT passed the INSERT with VALUE question.")
    <- .send(agMaterial, achieve, hasPrerequisite(insert, select)).

```

**Fig.5. 20:** Agent plans based on the derived FOL syntax specified in Chapter 4 for classification of student knowledge on the DELETE desired concept.

This set of rules can be explained further using the IF...THEN statement as condition-action rule as indicated in Russell & Norvig (2010) simple reflex agent. The *passed* or *failed* predicates of a FOL statement are categorical features for classification that is decided by the agent *agSupport*. All the agent *agModelling* does is to take the inputs and decide which of the number of classes (called *N classes* by Marsland, 2014) the students belongs to. Thus, if a set of percepts or input attributes are all *<passed>* (e.g. label *@d1*) then the student has *positive ability* to learn his desired concept, that is the *delete*. That is,

```

IF
    desired_Concept("delete")
    & passed("The student has passed the insert with select question")
    & passed("The student has passed insert with value question")
THEN
    Delete URL

```

But if the set of input is a mix of both *<Passed>* and *<failed>* (e.g. label @d2 then it is *partial ability*. The student learns the failed concept `insert_value`:

```

IF
    desired_Concept("delete")
    & passed("The student has passed the insert with select question")
    & failed("The student has NOT passed insert with value question")
THEN
    insert_value URL
    
```

But if the set is a mix of both *<failed>* and *<passed>* (e.g. label @d3) in reversed order to @d2, then it is also *partial ability*. The student learns the failed concept `insert_select`:

```

IF
    desired_Concept("delete")
    & failed("The student has NOT passed the insert with select question")
    & passed("The student has passed insert with value question")
THEN
    insert_select URL
    
```

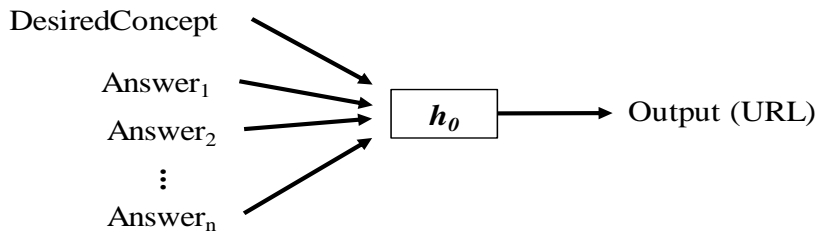
But if the set are all *<failed>* predicates (e.g. label @d4) then the student has *negative ability*. Then the student learns all the failed concept `insert_select`, and `insert_value` as shown below:

```

IF
    desired_Concept("delete")
    & failed("The student has passed the insert with select question")
    & failed("The student has NOT passed insert with value question")
THEN
    insert_select URL, insert_value URL
    
```

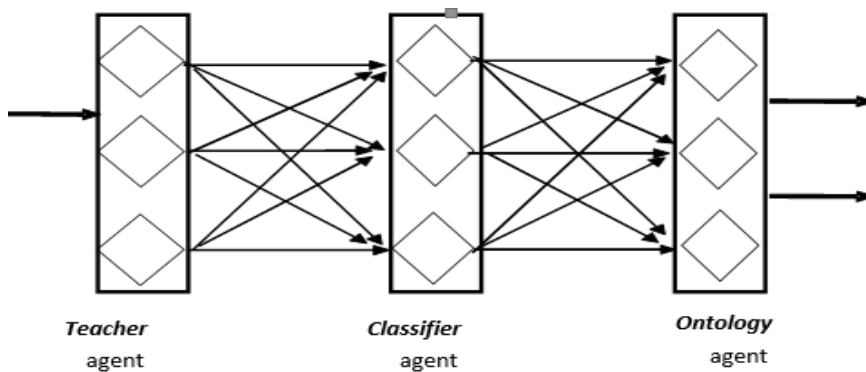
On the pre-assessment system, all the set of predicate in the *context* part of the agent plan corresponds to the student behaviour. Noticed that the parameter *<D>* is part of all the predicate clauses in the classification plan *context*. The parameter, as part of the decision clauses, identifies a student's desired concept as well as the prerequisite leaf nodes connected to the desired concept. In Jason, at the fulfilment of these conditions (the *ifs*), the *triggering\_event* is adopted for the execution of the plan *body*.

From the foregoing analysis, the process of the *Input-communication-classification* in the Pre-assessment System MAS is presented in Figure 5.21:



**Fig.5. 21:** Inputs, communication and classification in the multiagent Pre-assessment System. Inputs are serial, as students reaction to the System.

where the *communication-classification* stages are represented as  $h_0$  function that is further broken down into a serial or asynchronous process of communication between agents in Figure 5.22. This mirrors the *one-pass vertical architecture* (Chin *et al.* 2014) of agents such that the agent *agInterface* obtains the sensor input, communicate the input as messages through from agent to agent that all along the way performed their roles according to design, and finally to the effector agent that releases the URL links to the student. The three agents in Figure 5.22 are reactive agents with individualised plans represented in decision symbols: that represents agent plans that are triggered based on the percept received from incoming messages. The triggered plan is dependent on the plan *context* that is satisfied. The end of a pre-assessment session is at the time the ontology agent *agMaterial* releases learning material(s).



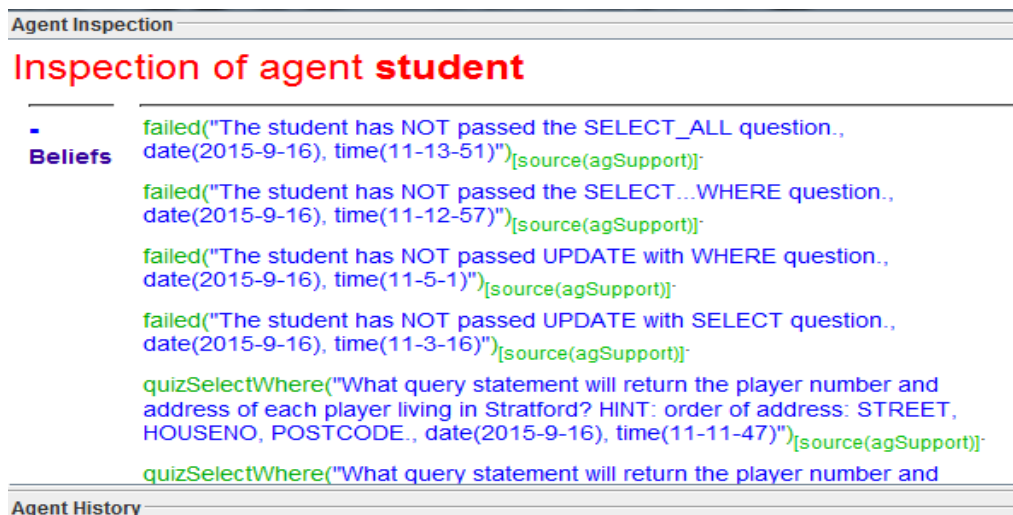
**Fig.5. 22:** One vs. All Multiple Classification (Ehimwenma, Beer & Crowther, 2016a)

Rules representations (plan *context*) are beliefs about the state of the world (student learning). In communication, the agents are reactive and they use deliberation as a

*means to an end*: Deliberation here, involves (usually systematic) exploration of alternative courses of action (Logan, 2014). The input becomes beliefs that are matched with pre-conditions for plan selection. The output of one agent behaviour becomes the input of another agent. In other words, there is a condition(s) match of the representation of current state to previous percept or message; and each agent output is a predicate statement to the next agent.

### 5.9.3 Agent *agModel* and Student History

The agent *agModel* is the *Student* agent. It is the agent that keeps track of the students' pre-assessment history. This history is comprised of the *desired concept* <*D*> and *answers* <*V*> to every question. This parameter information is also communicated by the agent *agSupport* after every pre-assessment activity and are persistently stored in the agent *agModel* text database using the Jason *TextPersistentBB* Class. The stored information is meant for the course tutor to monitor students' learning and their technical difficulties in their SQL query constructs. Figure 5.23 illustrates some of the information stored in the text database.



**Fig.5. 23:** A snapshot of the agent *agModel* (student) *Mind Inspection* of updated beliefs in Persistent beliefs after some pre-assessments by the MAS.



### 5.9.4 The Agent *agSupport* and Pre-assessment

This is the *teacher* that pre-assesses students using `!achievement` goals for questions retrieval from its beliefs. For instance, given that a desired concept is *update*, *agSupport* first enquires from the agent *agMaterial* whether the *update* concept exists in the ontology with the message (see Fig. 5.24):

```
.send(agMaterial, askOne, hasPrerequisite(V, delete));
```

The agent *agMaterial* replies back that the *Update* concept has prerequisite *delete*. The *askOne* performative message does not update the belief of a receiver agent. Instead, it triggers the agent *agMaterial* to reply to the sender with the content requested. On receipt of the replied message, the sender agent *agSupport* belief is updated with the new information. Based on the FOL logic information that is now available to the agent *agSupport*, it then informs the student that the concept entered has a prerequisite in the given code

```
.print(V, "hasPrerequisite delete");
```

Thereafter, *achievement goal*

...

```
!quizDeleteSelect(DeleteSelectQuiz).
+!quizDeleteSelect(DeleteSelectQuiz):quizDeleteSelect(DeleteSelectQuiz)
<- ...
```

as the next intention in the plan is adopted with the condition that the `quizDeleteSelect(DeleteSelectQuiz)` in the plan *contexts* exists in the agent *BB*, then the body of the plan is executed.

In the body of the plan, *date* and *time* are stamped to every activity of students. This is from the stage of the desired concept to the stage of the materials recommended for learning. The essence of this is to record time lapse on every event in order to make comparison with the outcome of pre-assessment. Then the desired concept is sent to the agent *agModelling* (the *classifier*). Afterwards, the quiz of the first or left most leaf node to the delete concept i.e. *deleteSelect* is released to the student (Fig. 5.24). As shown in the DL definition and in Figure 5.9, every leafnode has a corresponding question. On receipt of the quiz, the student enters his answer. The agent *agSupport* receives the answer from the agent *agInterface*, and sends an answer to agent

*agModelling*. At this stage the student is assessed on the answer and informed of the outcome.

```

...
.send(agMaterial, askOne, hasPrerequisite(V, delete)); //Asking if relation
    exists in ontology
.println(V, " has prerequisite DELETE"); //action after getting reply
-value(V); //belief drop
.println("Question on DELETE with SELECT:");
.println;
!quizDeleteSelect (DeleteSelectQuiz) .

+!quizDeleteSelect (DeleteSelectQuiz) : quizDeleteSelect (DeleteSelectQuiz)
    <-.date(YY, MM, DD);
    .time(HH, NN, SS);
    .println(DeleteSelectQuiz);
    .concat(DeleteSelectQuiz, ", date(",YY,"-", MM,"-", DD, ") ", " ", " ",
    "time(",HH, "-", NN, "-", SS, ") ", Qds);
    .send(student, tell, quizDeleteSelect(Qds));
    .println;
    .wait(6000000).

```

**Fig.5. 24:** Agent achievement goal for retrieving and displaying the deleteSelect quiz from BB.

For a passed assessment, this the plan behaviour of the agent assessment, feedback and communication of the decision process (Fig. 5.25). The agent takes decisions on the answers received from *agInterface* and communicate the *passed* or *failed* decisions statements, including feedbacks to students. Thereafter the quiz of the next leaf node of the delete concept i.e. *deleteWhere* is released by agent *agSupport* through the adoption of the next agent *achievement* goal. In the process of pre-assessment, the agent *agSupport* uses *achievement* goals within plans to navigate from question to question in its beliefs. At every stage of pre-assessment, the agents *agModelling* (*classifier*) and *agModel* (*or student*) are directly communicated (*see Fig.5.25*). This implementation has been with two leaf nodes per class node based on the principle of *Chunking* (Casteel, 1988; Anderson, 2008).

```

@p16
// Plan for correct answer to DELETE_SELECT the first prerequisite to UPDATE.

+value(V) [source(agInterface)] : value(V) == value("DELETE FROM TENNIS_PLAYERS
WHERE TOWN = (SELECT TOWN FROM TENNIS_PLAYERS WHERE PLAYERNO = 44 AND PLAYERNO <>
44)") & testCount(0)
  <- .date(YY, MM, DD); .time(HH, NN, SS);
  .println("Good. Your answer is correct.");
  ?testCount(Count); -+testCount(Count+ 1);
  .concat(V, ", date(", YY, "-", MM, "-", DD, ")", ", ", "time(", HH, "-", NN, "-",
  " SS, ")", Rds1);
  .send(student, tell, responseToDeleteSelect(Rds1)); //date and time appended
  PassedDS = "The student passed DELETE with SELECT question.";
  .concat(PassedDS, ", date(", YY, "-", MM, "-", DD, ")", ", ", "time(", HH, "-",
  " NN, "-", SS, ")", Pds);
  .send(student, tell, passed(Pds));
  .send(agModelling, tell, passed("The student passed the DELETE with SELECT
  question."));
  .println("Question on DELETE with WHERE clause:");
  !quizDeleteWhere(DeleteWhereQuiz); .println.

+!quizDeleteWhere(DeleteWhereQuiz) : quizDeleteWhere(DeleteWhereQuiz)
<- .date(YY, MM, DD); .time(HH, NN, SS);
  .concat(DeleteWhereQuiz, ", date(", YY, "-", MM, "-", DD, ")", ", ",
  "time(", HH, "-", NN, "-", SS, ")", Qdw);
  .send(student, tell, quizDeleteWhere(Qdw)); //date and time appended
  .wait(6000000); .println.

```

**Fig.5. 25:** Plan snapshot for a passed answer assessment, user feedback, communication and next quiz display use of achievement goal by the agent agSupport

### 5.9.5 Agent *agMaterial* and Ontology

This is the agent that has the SQL ontological relation initialised as internal knowledge beliefs in FOL *ground* facts. The agent take message percept, matches the concepts in every relation as requested and *directed*, and retrieves the information or literal from its BB. For example, an *askOne* request from the agent *agSupport* that confirms a student's desired concept when submitted at the interface. The agent holds the learning materials in their URL (universal resource locator). At the end of a pre-assessment session, the agent makes URL(s) available to students by matching a plan *context* to the *achieve* performative message as directed (a *directive*, Searle, 1959) by the

classifier agent—after the student is classified. An *askOne* performative from agent *agSupport* and the *achieve* performative from agent *agModelling* is an order that commits the agent *agMaterial* to the message content. The content of these performatives were successfully executed by the agent *agMaterial*. In the agent beliefs, *ground* facts are represented in FOL as:

- class to class with *hasPrerequisite* relation;
- class to leaf nodes (subclass) with *hasKB* relation;
- leaf node to data values with *hasContent* relation;
- class to class with *isPrerequisiteOf* relation

as defined in the SQL TBox.

The properties *hasPrerequisite* and *hasKB* relations are the ObjectProperty, and the *hasContent* a DataProperty as in Protégé (Horridge *et al.* 2004). The Figure 5.26 present a snapshot of a plan with the *hasKB* predicate e.g.

```
+!has_KB(delete, deleteSelect)
```

that is adopted by the agent *agMaterial* when the sending agent *agModelling* has concluded classification. Every plan in the agent *agMaterial* is for recommendation of learning content to direct a suitable level(s) of learning material for student.

```
@u_m3
+!has_KB(delete, delete_select)[source(agModelling)] // for failure of the
DELETE_SELECT of desired_Concept("UPDATE")
<- .println(" You will learn the DELETE_SELECT. Please use the text link below:");
   ?hasContentText(deleteSelect, DS_textURL)[o(sql)];
   .println("DELETE...SELECT query Text Link: ");
   .println(DS_textURL).
```

**Fig.5. 26:** Adoption of a *hasKB* predicate relation, and content query from BB with *?hasContent* test goal in a plan.

The agent *agModelling* uses the *hasPrerequisite* or *hasKB* predicate in its message At the receipt and adoption of the plan with this message as the triggering event, the agent *agMaterial* uses a test goal given in the form

*?hasContent(x, y)*

to query its BB for the release of learning material. The *hasContent* data property relation is suffixed with *Text*, such as:

```
?hasContentText(updateWhere, UW_textURL)[o(sql)];
```

to depicts the type of learning material on the URL links.

## 5.10 Summary of Chapter

One of the objectives of this system is to unravel gaps in students learning and to adequately support them to fill-in the gaps. The failure of any prerequisite concept when a student intends to learn a top or higher concept means a gap in his learning.

This Chapter has presented the implementation of the Pre-assessment System and its SQL ontology learning structure towards the objective of identifying gaps in learning. Given Maedche & Staab (2001) 5-tuple [C, R, F, A, I], the SQL ontology was defined using formal concepts. Firstly, the SQL ontology was defined with a description logic TBox terminology and ABox assertion. While the TBox described the terms and relations in the SQL domain ontology, the ABox asserted the individual members. The terms in the TBox were analysed and different ontology models were constructed given the role (or relation), the constraints and the minimum cardinality of  $\geq 2$  specified for leaf nodes. But since learning is sequential, the linear model was adopted for implementation. The linear model has a *regular* model as well as a *non-regular* ontology model. In furtherance, the chapter demonstrated the classes and relations using the Jena API ontology model and Protégé ontology illustrations, and then parse the Protégé OWL ontology in Jena (an RDF API) to observe: 1) the OWL class to class relation, 2) OWL class to rdfs subclass relation, 3) the object properties that exists between *rdfs domain* and *range* in TURTLE syntax in order to capture OWL expressiveness over RDF(S). TURTLE outputs ontology listings in *concepts'* given names, and not in their fully qualified URI namespaces such as in RDF/OWL or OWL/XML syntax. Based on *concepts'* given names and their property, first-order logic (FOL) representation was used to specify agent beliefs or *ground* facts in a system that has been implemented in Jason AOP. The chapter then presented the Pre-assessment System, and its detailed structure as specified with the PDT AUML tool in Chapter 4. This covered the agents, their functions or role in the system, CArtaGO and percept observation, agent localised or internal knowledge base in FOL, and inter-

agent communication of ontological knowledge. As presented in Chapter 4, two strategies of pre-assessment were identified given the Pre-assessment Mechanism. This chapter has implemented and tested the strategy of *pre-assessment by immediate prerequisite class* and its classification process. While the results of this implementation and evaluation shall be presented in Chapter 6, details of the *pre-assessment by multiple prerequisite classes* (the second strategy) shall be presented in Chapter 7.

# Chapter 6

## System Evaluation, Results and Analysis of Data

### 6. Introduction

Chapter 5 started by introducing the learning structure of the SQL domain of this thesis. Using a description logic language, the concepts of the SQL ontology and inter-concept relationships was defined with a minimum cardinality specification of two leafnodes per parent class. From the various ontology model analysis given the TBox definition, this research adopted the linear model as the optimum model for implementation on the Pre-assessment System. This is to allow students to progress gradually from one level of pre-learning to the next without missing any concept. Based on the linear model, beliefs or facts representation in first-order logic (FOL) and speech acts (performatives) based inter-agent communication in the Pre-assessment System was implemented using Jason AgentSpeak language. Afterwards, the System was evaluated for fitness-of-purpose, which is, to *identify gaps in students' learning*. Thus, this Chapter 6 presents the evaluation of the Pre-assessment System, the data collected and the analysis of the data. This includes students' skills data and their experiential feedback after their pre-assessment exercise. From the results, the data on students' real-time engagement with the Pre-assessment System reflects students' understanding of SQL queries. In the post pre-assessment data which is qualitative, students expressed their thoughts through questionnaire that was administered via the SurveyMonkey (2017).

## 6.1 Sampling Technique

This section presents the process of sampling in the survey and the collection of data in the research.

- **Population:** The population of the study is SQL/database students. This is because the content of learning of the Pre-assessment System is SQL. With the identified population sample, the system can be effectively evaluated for fitness of purpose and results validation given that the population are participants in the learning domain.
- **Sampling Frame:** The sampling frame are database students of the Sheffield Hallam University. The is comprised of students that are in their first year undergraduate, second year undergraduate course through to Master's degree level. They are students that have either studied database modules in their recent past or in their current learning.
- **Sampling Method:** The method of sampling used for the chosen population is the random sampling technique. Firstly, after consulting with the lecturers in charge of the databases courses, emails were then sent out via the Sheffield Hallam University Blackboard site to request for volunteer participants in the study. Apart from the use of emails, the course lecturers also candidly announced in the classrooms to remind students of participation. Due to the imbalance of demographic representation such as ethnicity in the database modules, demographic data was later dropped for consideration in the study.
- **Sample Size:** All the students who volunteered for the study also took part in the survey which is about the *identification of learning gaps* in students' SQL query skills. The sample size of 7 students that volunteered for the survey and their course distribution in a survey that was conducted over four academic semesters is shown in TABLE 6.1.



**TABLE 6. 1: SAMPLE SIZE OF VOLUNTEERS AND RECRUITMENT RECORDS**

S/N	Semester/Academic Year	No. of Participants
1.	Semester 1, 2014/15	2
2.	Semester 2, 2014/15	2
3.	Semester 1, 2015/16	0
4.	Semester 1, 2016/17	3
	<b>TOTAL</b>	7

## 6.2 Experimental Setup

This section presents the different stages of the Pre-assessment System's evaluation exercise and the data collated in tables after analysis.

### 6.2.1 Recruitment for Evaluation Exercise

SQL is one of the technical fields of programming in computing science. It can be tricky to learn and easily forgotten when learned. As described in Chapter 2, the skills in SQL are challenging and students have many difficulties learning them (Mitrovic 1998). In Prior (2003) it was ascertained after their experimentation that the learning and mastering of these (SQL) skills is a difficult process that requires considerable amount of practice and effort on the part of students. Prior (2003) stated is not easy for students. Therefore, to ease the difficulty in the learning of SQL, strategies that supports the best learning practice was considered. This further informed the choice of our linear ontology models of SQL concepts implementation in batches (*chunks*) and class by class in a simple-to-complex order. This is to model learning path and resource for students to succeed.

So having developed the System to test SQL previous knowledge gaps or gains, sessions were organised for testing the focus group—computing students that have taken modules in Databases. As students that have previous knowledge of SQL, it was

believed that students have the capability to holistically evaluate the system to address their learning needs in the domain of SQL. With the necessary requirements of the *Research Ethics standards* met, calls for volunteer-participants were made for the evaluation of the system to:

- Pre-assess students' skills in the domain of SQL (the context in which the system has been developed).
- Evaluate the system's fitness for purpose i.e. test of the underlying pre-assessment mechanism, accurate classification, inter-agent communication and overall system design goal.

### **6.2.2 Student Consent and Lesson Plan**

As part of standard *Research Ethics* procedure, a *Consent Form* was designed for the study in order to obtain the participating students' consent (*see Appendix B, B.2 for consent form*). As a duly conceived teaching-learning session, a *Lecture Plan* was also designed. This was to guide students through their pre-assessment exercise.

Students were acquainted at the beginning of the pre-assessment sessions with the objectives of the test exercise—which was to identify gaps in previously learned SQL knowledge. Students were informed that the session was not a formal faculty examination. Rather it was a research survey of a multi-agent Based SQL Pre-assessment System developed to assist the learning of SQL. As such there was the need to have some independent body (like them — students in Databases or SQL) that could evaluate the system's function or performance, and then make feedback to the researcher. The essence is to support the learning and teaching of SQL. In doing so, that their personal data or information obtained would not be divulged in any form.

In addition, the students were informed that, by no means, were they compelled to participate in the exercise. They could accept to continue or opt out of the research exercise at any moment. However, their participation in the evaluation exercise was highly solicited and important to the study. On those grounds, the students gave and signed their Consent, and the Lecture Plan were handed out to them for the commencement of their pre-assessment exercise.

Furthermore, it was explained that the objective of the system was to find out whether gaps exist in their SQL knowledge. That when they [students] enter a topic (among a

list of topics on the system) that they intend to learn, the system would present to them some questions on the prerequisites to the topic that was entered: To ascertain whether the students are ready for the new topic they intended to learn or whether there are previously learned modules that needed to be revisited. Finally, that, while they would engage the Pre-assessment System, the answers that were provided would be logged in the system for the researchers' review.

### 6.3 Pre-assessment Skills Data Collection and Analysis

The pre-assessment exercise took place in different academic sessions as shown in TABLE 6.1. As students worked on the system they equally got feedback from the System, their correct query constructs were adjudged as *passed* and the incorrect ones as *not passed* (i.e. *failed*).

Recall that in Chapters 4 and 5, the pre-assessment System also keep the history of students' activities. Thus the following are examples of the pre-assessed data stored permanently by the agent *agModel* (student) in the system (*complete data in Appendix A, A.1*):

- **Example Data 1**

The

```
desired_Concept("INSERT, date(2017-1-26), time(12-10-23)") [source(agSupport)].
```

is the *INSERT* desired concept entered by the student, and

```
quizSelectWhere("What query statement will return the player number and address of each player living in Stratford? HINT: order of address: STREET, HOUSENO, POSTCODE., date(2017-1-26), time(12-10-23)") [source(agSupport)].
```

the quiz of *SELECT\_WHERE*, the first leaf node prerequisite to *INSERT*; and

```
responseToSelectWhere("SELECT PLAYERNO, STREET, HOUSENO,  
POSTCODE, date(2017-1-26), time(12-13-  
54)") [source(agSupport)].
```

the student response to the quiz of `SELECT_WHERE`, then

```
failed("The student has NOT passed the SELECT...WHERE  
question., date(2017-1-26), time(12-13-  
4)") [source(agSupport)].
```

which is the failed predicate decision statement after assessment by the agent *agSupport*. The message that is also sent to the agent *agModelling* (classifier). This message is followed by the next quiz

```
quizSelectAll("State the SQL query that will output all the  
data in TENNIS_TEAMS?, date(2017-1-26), time(12-13-  
54)") [source(agSupport)].
```

is the quiz of `SELECT_ALL`, the second leaf node prerequisite to `INSERT`. Then

```
responseToSelectAll("SELECT PLAYERNO, STREET, HOUSENO,  
POSTCODE, date(2017-1-26), time(12-13-  
59)") [source(agSupport)].
```

which is the student response to the quiz of `SELECT_ALL`, and then the

```
failed("The student has NOT passed the SELECT_ALL question.,  
date(2017-1-26), time(12-13-59)") [source(agSupport)].
```

which is the *failed* predicate decision statement that is also a message sent to the agent *agModelling* (classifier).

After accumulating the two *failed* predicate decision statements, the agent *agModelling* (classifier) classified the student for learning by sending an *achieve* performative message to the agent *agMaterial* as specified with the Prometheus PDT design tool in Chapter 4. The agent *agModelling* (classifier) does this by matching the

message content in their unary logic form to its array of plans, and triggering the plan whose plan context is selected before communicating the agent *agMaterial* to release the web URL link. This, the student placed on a browser to study the two *failed* concepts in this case.

▪ **Example Data 2**

In this pre-assessment,

```
desired_Concept("UNION, date(2017-1-26), time(12-42-14)") [source(agSupport)].
```

is the *UNION* desired concept entered by a student, and

```
quizFullOuterJoin("Give, for each player, the player number, the name and the penalties incurred by him or her; order the result by player number. (HINT: you need to use OUTER JOIN), date(2017-1-26), time(12-42-14)") [source(agSupport)].
```

the quiz of *FULL OUTER JOIN*, the first leaf node prerequisite to *UNION*; and

```
responseToFullOuterJoin("SELECT P.PLAYERNO, P.NAME, PEN.AMOUNT, date(2017-1-26), time(12-59-10)") [source(agSupport)].
```

the student response to the quiz of *FULL OUTER JOIN*, then

```
failed("The student has NOT passed the FULL OUTER JOIN question., date(2017-1-26), time(12-59-10)") [source(agSupport)].
```

which is the *failed* predicate decision statement taken and as the message that is sent to the agent *agModelling* (classifier). Then the next quiz

```
quizInnerJoin("For each player born after June 1920, find the name and the penalty incurred by him or her? HINT: you need to use INNER JOIN, date(2017-1-26), time(12-59-10)") [source(agSupport)].
```

is the quiz of INNER\_JOIN which is the second leaf node prerequisite to UNION.

Then

```
responseToInnerJoin("SELECT P.PLAYERNO, P.NAME, PEN.AMOUNT  
FROM TENNIS_PLAYERS P INNER JOIN TENNIS_PENALTIES PEN ON  
P.PLAYERNO = PEN.PLAYERNO, date(2017-1-26), time(13-1-  
19)") [source(agSupport)].
```

which is the student response to INNER\_JOIN, and then the

```
passed("The student has NOT passed the INNER_JOIN question.,  
date(2017-1-26), time(13-1-19)") [source(agSupport)].
```

which is the *passed* predicate decision statement which is also a message to the agent *agModelling* (classifier). In this pre-assessment, the student only *failed* one prerequisite. Thus, the student was recommended to the Full\_Outer\_Join URL link being the *failed* concept.

- **Example Data 3**

In contrast to *Example 1* and *Example 2* above, in *Example 3*, the two leafnode prerequisites to the INSERT was *passed* by the student when

```
desired_Concept("INSERT, date(2015-10-16), time(11-11-  
47)") [source(agSupport)].
```

INSERT was entered as the desired concept. The prerequisite quiz

```
quizSelectWhere("What query statement will return the player  
number and address of each player living in Stratford? HINT:  
order of address: STREET, HOUSENO, POSTCODE., date(2015-10-  
16), time(11-11-47)") [source(agSupport)].
```

of SELECT\_WHERE was displayed. The

```
responseToSelectWhere("SELECT STREET, HOUSENO, POSTCODE FROM  
TENNIS_PLAYERS WHERE TOWN="Stratford";, date(2015-10-16),  
time(11-12-57)") [source(agSupport)].
```

was the response from the student. Then the student was assessed to have *passed*

```
passed("The student has passed the SELECT...WHERE question.,  
date(2015-10-16), time(11-12-57)") [source(agSupport)].
```

Then the next quiz

```
quizSelectAll("State the SQL query that will output all the  
data in TENNIS_TEAMS?, date(2015-10-16), time(11-12-  
57)") [source(agSupport)].
```

of the SELECT\_ALL statement was released, and the student responded with

```
responseToSelectAll("SELECT * FROM TENNIS_TEAMS;, date(2015-  
10-16), time(11-13-51)") [source(agSupport)].
```

which is the correct answer to SELECT\_ALL, and the student was also assessed to have

```
passed("The student has passed the SELECT_ALL question.,  
date(2015-10-16), time(11-13-51)") [source(agSupport)].
```

the SELECT\_ALL prerequisite leafnode quiz. In this case, the student was recommended to learn the desired concept having *passed* the prerequisite quizzes.

▪ ***Example Data 4***

There were occasions after a desired concept was entered and quiz released, because students spent their time trying to work out their query statements, the system clocked out. An example is,

```
desired_Concept("INSERT, date(2015-10-16), time(11-8-  
32)") [source(agSupport)].
```

then the quiz

```
quizSelectWhere("What query statement will return the player
number and address of each player living in Stratford? HINT:
order of address: STREET, HOUSENO, POSTCODE., date(2015-10-
16), time(11-8-32)") [source(agSupport)].
```

that was not responded to. In such cases, students had to restart the MAS. For the complete data set that was stored in the agent *agModel* belief base (see Appendix A, A.1). The TABLE 6.2 presents the data of the number of correct answers and that of the incorrect answers entered in the system by all 7 participants who took part in the survey.

**TABLE 6. 2: PERCENTAGE OF CORRECT AND INCORRECT PRE-ASSESSMENT ANSWERS**

<i>No of Students</i>	<i>Percentage (%) Correct</i>	<i>Percentage (%) Incorrect</i>
7	22.7%	77.3%

In the TABLE 6.2 a total of **22.7%** (*passed*) correct answers were entered for queries as against incorrectly answered queries **77.3%** (*failed*) pre-assessments, respectively; (see Chapter 7 for breakdown).

### 6.4 Post Evaluation and Experiential Feedback Data

To gather students’ perception about their user experience on the Pre-assessment System, a post-evaluation survey was conducted through a 17 item questionnaire. The questionnaire was designed by the researcher, and was vetted and validated by the supervisory team as suitably adequate for the collection of the relevant data with respect to the system’s design and the SQL domain of learning. The questionnaire contained both structured and unstructured items with 11 structured items that can be ticked, and 6 unstructured items of open-ended entries that requires short textual response. The TABLE 6.5 contains the structured data of 11 items, while the Tables



6.3, 6.4 and 6.6 – 6.9 have the unstructured data entries as obtained from the administered questionnaires via SurveyMonkey (2017).

**TABLE 6. 3: QUESTION 1. COURSE OF STUDY?**

<i>Course</i>	<i>Percentage (%)</i>
BEng (Hons) Software Engineering	29%
MSc Database Professional	14%
Enterprise System Professional	14%
BSc Info Tech with Business Studies	43%
<b>Total</b>	<b>100%</b>

**TABLE 6. 4: QUESTION 2. YEAR OF STUDY?**

<i>Year</i>	<i>Percentage (%)</i>
First Year	14.3%
Second Year	71.4%
Masters	14.3%
<b>Total</b>	<b>100%</b>

**TABLE 6. 5: QUESTIONS 3 – 13**

<i>Questions (Q)</i>	<i>Strongly agreed</i>	<i>Agreed</i>	<i>Undecided</i>	<i>Disagreed</i>	<i>Strongly disagreed</i>
<b>Q3:</b> The system was useful	14.29%	71.43%	14.29%		
<b>Q4:</b> The system helped me to recall my previous knowledge	42.86%	57.14%			
<b>Q5:</b> The system supports the learning of SQL	28.57%	57.14%	14.29%		
<b>Q6:</b> I am not familiar with SQL	14.29%			57.14%	28.57%
<b>Q7:</b> The system provided guidance to learning materials		85.71%	14.29%		
<b>Q8:</b> The system has a use-able interface		57.14%	14.29%	28.57%	

<b>Q9:</b> I understood the purpose of the system	42.86%	57.14%			
<b>Q10:</b> The tutor was helpful in introducing the system	57.14%	42.86%			
<b>Q11:</b> The tutor was helpful in providing assistance	57.14%	42.86%			
<b>Q12:</b> The session's organisation was a good learning experience	14.29%	57.14%	14.29%	14.29%	
<b>Q13:</b> The session was well organised	28.57%	57.14%	14.29%		

The following Tables 6.6 – 6.9 presents the open-ended responses from participants of the Pre-assessment System and the pre-assessment sessions:

**TABLE 6. 6: QUESTION 14. WHAT WAS MOST INTERESTING ABOUT THE SESSION'S ORGANISATION?**

#	Responses
1	Easy to understand and work with. Well developed and presented.
2	It was interesting to see what the design of the program was like and how it could be used in education
3	Learning about this new system and how it can help teach people how to use SQL.
4	Layout of the testing session we well organised, testing environment was good.
5	Learning that there are multiple ways in which SQL can be learnt and people are working on alternate methods.
6	It is actually have good objectives, so we will learn what exactly we need to learn. Because sometimes tutor teach something which is redundant since some people already understand it well.
7	Support and proper guidance through the whole process.

**TABLE 6. 7: QUESTION 15. WHAT WAS LEAST INTERESTING ABOUT THE SESSION'S ORGANISATION?**

#	Responses
1	Lack of equipment available. Session was slow.
2	Just waiting to have a go
3	N/A
4	Nothing
5	We only had one monitor to do the work on.
6	This system is not quite flexible and does not allow trial and error terms. One small error led into decision that we need to learn the module. Meanwhile, it supposed to be more flexible in programmin language because to get one result, we will have various way to achieve it.
7	NA

**TABLE 6. 8: QUESTION 16. WHAT IS MOST INTERESTING ABOUT THE SQL SYSTEM?**

#	Responses
1	Seeing how the software enables students to progress through the different sections only when they have successfully mastered the previous level.
2	The given links for help
3	There were different websites that could be referred to to help teach people about SQL and help them understand how to construct queries.
4	The information links at the end based on the user input.
5	The system provides students with multiple topic areas and eases you into the code.
6	SQL is an open source knowledge and does not really need to remember many things, just need to remember the commands and we can explore to fix the problem independently.
7	The concept is an interesting one

**TABLE 6. 9: QUESTION 17. WHAT WAS LEAST INTERESTING ABOUT THE SQL SYSTEM?**

#	Responses
1	Having to switch between three different windows to operate the system.
2	I found the concept a little confusing at first because the tables are already there so I didn't understand what would need inserting without being given data and the interface wasn't very easy to easy
3	N/A
4	Maybe the user interface.
5	The system only covers limited SQL statements so when more are added I think it will be more interesting.
6	Nothing least interesting about SQL.
7	NA

## 6.5 Summary of Chapter

The Pre-assessment System has been evaluated, and data was collected in this chapter. The data collected from a small sample size of 7 database students was presented. The sample size is the number of participants that volunteered to partake in the survey. Of no doubt, participant recruitment for the study has been a challenge. Nonetheless, from the available sample size and system evaluation, it is found that the system has been able to identify gaps in students' SQL query constructs. This is on the strategy of *Pre-assessment by Immediate Prerequisite Class* using a *regular ontology* model of two leaf nodes to a class node (Chapter 5, Fig. 5.3) that was implemented. The chapter also presented the pre-assessment data and showed how students were pre-assessed as the System navigated from one leaf node concept to another underneath their desired concept. Altogether, the data collected and analysed reflects students' know-how of SQL query skills, quantitative and as well as qualitative data analysis. From the SQL knowledge or skills related data, the difficulty faced by a cross section of students have been unravelled. This can enable the course tutor to meet the learning needs of students. This knowledge data as presented conforms to Prior (2003) assertion that SQL is not easy to learn and that students are faced with challenges and difficulties in writing SQL queries. At the end of the pre-assessment sessions, open ended views were collected as feedback from students via *SurveyMonkey*. This was for the elicitation of facts about their user experience. In next Chapter 7, further discussion is presented about the pre-assessment data, and its implications for the teaching of SQL. Also discussed is the strategy of *Pre-assessment by Multiple Prerequisite Classes* as well the process involved in the development and operations of the Pre-assessment System.

# Chapter 7

## Discussions

### 7. Introduction

The aim of this research was to identify gaps in students' learning in order to provide assistance in filling those gaps by pointing students to the materials of the concepts or unit of lessons that they needed to know. To that effect, the agent based Pre-assessment System was proposed and developed to use a classification approach that can categorise students' skills and recommend materials that would help to close the gaps in students' learning.

In a formal school curriculum i.e. universities, schools (e.g. Manouselis *et al.* 2011), learning is sequential and ordered from *known* (learned concepts) to the *unknown* (higher concepts). As a *formative* type (Conole & Warburton, 2005) of prior knowledge assessment system, the Pre-assessment System has its concept of learning structured in an ordered sequence. In this arrangement, diagnosis of students' understanding of prior SQL domain concepts is carried out so that support can be provided for further learning through the planned pre-assessment strategies earlier described in Chapters 4.

### 7.1 Dealing with The Research Question

The purpose of this research was to identify gaps in students' learning: between a target learning concept of the student (a higher concept) called the "*desired\_Concept*" and some previously learned concepts (the lower level concept). To achieve this aim, a research question RQ was formulated towards the development and realisation of a formative type of assessment system as:

*How can students be helped to identify gaps in their current learning so that they can be fully prepared for the next stage in their learning?*

The approach to answering this RQ has been through: the development of the Pre-assessment System, evaluation of the system, and the collection of students' activities and skills data from the *agent agModel* persistent beliefs after the agent *Mind inspection* (Bordini, Hubner & Wooldridge, 2007). Agent *Mind inspection* is a view into an agent belief update by the programmer or researcher, *see Chapter 5, Figure 5.3*.

### 7.1.1 How System Identified Gaps and Material Recommendation

The System has helped students to self-diagnose their SQL skills. This has been through a process in which students are prompted to enter a *desired\_Concept* from a hierarchy of SQL class concepts or topics (*See Figure 5.19*). Thereafter, pre-assessment on some prerequisite leafnodes to their chosen concept is carried out. This is because every student cannot start in the same learning block, as such, there has to be a different choice-levels of pre-assessments. While a student may desire to study a higher concept, the research wanted to ascertain whether the student has a good knowledge of prerequisites to the *desired\_Concept*. In that perspective, pre-assessment or pre-learning diagnosis needs to take students from one lower-level to the next higher-level concept after assessment. This is when students have demonstrated an appropriate level of skills at the lower level. On one hand, this is similar to the strategy used in the PAT Algebra System (Ritter *et al.* 1998) that promote students to a higher level-learning after completing a task at a lower level. In contrast to the PAT Algebra System and also a number of SQL systems that provides tutorials e.g. "SQLCourse.com" (*see Chapter 2*), but not assistance for errors, the Pre-assessment System makes material recommendation for the learning of unlearned i.e. the *failed* concepts after pre-assessment. The act of making recommendations for the learning of *failed* concepts makes the Pre-assessment System different from the systems identified in literature (*see Chapter 2*) by the strategies of pre-assessment and classification employed in this thesis.

As presented in Chapter 6, the Pre-assessment System evaluated students' skills prior to learning a higher or *desired\_Concept*. During pre-assessment sessions, as prerequisite questions were presented to the participants (students) in the study,

students responded by entering SQL answer queries from question to question: questions that corresponded to the prerequisite class concepts whose leafnodes  $N$  have been defined to have  $N \geq 2$  minimal cardinality in the TBox, see Chapter 5, Figure 5.2. As described in Chapter 5, implementation of an ontology of learning concepts in the Pre-assessment System can be of at least leafnodes  $N = 2$  per parent class which has been implemented and evaluated, and of leafnodes  $N \geq 2$  per parent class implementation that is presented in this chapter.

While student participants engaged with the System, the System continuously interacted with students, informing them of the questions they have answered correctly or incorrectly. From the assessment on incorrect answers, students were able to identify their own learning gaps. After pre-assessment exercises, some students realised they were not ready for their higher and intended *desired\_Concept*. At the end of each pre-assessment exercise in which students were classified based on their skills, learning material URLs were presented, and students viewed materials on the web that provided assistance for their learning: That way the system provided assistance to students to close their learning gaps.

The Pre-assessment System is one that has been developed to be adaptable to students' level of learning of SQL. As stated in Michalski, Carbonell & Mitchell (2013) the level of adaptability provided by a system should be that which must present learning materials suitable to the state of knowledge of the student. Thus, the materials that were presented to students after their pre-assessments were tailored by the System to either the leafnodes of the *desired\_Concept* they intended to learn or to the *failed* leafnode(s) of the prerequisite concepts as defined in Chapter 4. See sections 4.7.1 and 4.7.2 for the FOL rules definition. The learning materials for a *desired\_Concept* were provided when a student *passed* all prerequisite questions considered and programmed under the *desired\_Concept*.

### 7.1.2 Initial System Development Stages

The Pre-assessment System has been developed using Jason AgentSpeak language, a first order logic (FOL) based language. During the early system developmental stages,

questions that Zhang, Kendall & Jiang (2002) described when developing an agent based system arose, namely: *what agent does what, what agent interacts, and how?* By further decomposing the aforementioned steps, subsequent questions ensued:

- What is the MAS going to observe?
- How will it observe?
- How will the MAS make decisions?
- How will it assist students to close the gaps in their learning?
- How and in what performative can agent communicate messages to understandably fulfil the goal of pre-assessment, *see Chapter 4, Figure 4.2.*

As described in Chapter 4, the approach is that the MAS observe a student's *desired\_Concepts*, present leafnode prerequisite questions and receive answer responses to the leafnodes prerequisite questions. The means, with which, this was done was through the CArtaGo artifact.

Jason AOP is language where beliefs representation and message content are in FOL. Given the beliefs in belief base (BB), agents make decisions by selecting the plan whose plan *context* matches the beliefs in their FOL representation. As stated in Chapter 3, Jason agent plan structure is of the form

*triggering\_event-condition-action.*

When the *condition* part of a plan is satisfied after some percept or accumulated messages in beliefs, the *triggering\_event* is adopted and the *action(s)* in the plan *body* is executed.

## 7.2 Reactive System

In Chapter 5, the Pre-assessment System was described as a system of five agents that is holistically a reactive system. This is because each agent reacts to perceived input(s) at appropriate triggering of an event. The agent *agInterface* can be referred to as the first reactive layer as it is the agent that observes the CArtaGo artifact. This is followed by others i.e. agents *agModelling*, *agSupport*, and *agMaterial* that takes individual decisions based on their individual plans and expected percepts. The agent



*agModel(student)* is the only agent whose function is to receive and keep persistent beliefs of all activities.

### 7.2.1 Agent Long term and Short term memory

Agents can possess both *long-term* and *short-term memory*. While the modelled facts that are initialised as beliefs in the agent is long-term memory, the updated knowledge as a result of inter-agent messages, can be said to be the short-term memory. As a reactive system, the short-term beliefs is the knowledge from which the agent recognises, matches and unifies with the long-term beliefs to perform a designated task. In convention as with volatile storage, agents' short-term beliefs are ephemeral or short-lived: They are lost when the MAS system is *Stopped*. The long-term belief is the agent permanent store that keeps updated beliefs, this beliefs or text knowledge base uses the *TextPersistentBB* class to keep track of all student activities during pre-assessment.

## 7.3 Agents Communication in The Pre-assessment System

In the Pre-assessment System, the essence of communication is for the agents to cooperate in the process of identifying gaps in students' learning and to assist in filling the gaps. In communication, there is a *sender* and a *hearer*, and the content of communication i.e. the message (Searle, 1969, Wooldridge, 2002, Labrou & Finin, 1998). Starting from the student user of the system down to all the agents of the Pre-assessment System, communication precedes reaction. Within the SQL Pre-assessment MAS, agents have engaged in communicative actions in order to share or transfer knowledge. This is carried out through *speech acts performatives* (Searle, 1969) in agent plans. Examples of the performatives in Jason AOP for developing the Pre-assessment System are *tell*, *achieve*, and *askOne*.

In the Pre-assessment System, agents communicate both unary literal in the form of  $p(a)$ , such as

```
value (V)
desired_Concept (V)
```

where V is the percept from environment, and also with binary literals  $p(a, b)$ , such as

```
hasPrerequisite(X, insert)
```

where agents mapped variables in their predicate statement using the *predicate* and a variable in a unary representation e.g. *desired\_Concept(V)*, or a predicate and one named literal in the statement e.g. *hasPrerequisite(X, insert)* in a binary representation. Based on the problem being addressed in this research that comprises the strategy of learning and understanding some lower concepts of a SQL domain before progressing to a higher class concept. The *hasPrerequisite* and *hasKB* are the predicates used for the set of semantic communications of facts between agents. While the *hasPrerequisite* is a link to individuals from a “*domain*” to a “*range*” (Horridge *et al.* 2004), the *isPrerequisiteOf* is the inverse relation from a *range* to a *domain* individual. As part of, for example, the *agent agMaterial* action, when the representation *hasPrerequisite(high\_concept, low\_concept)* is received, the agent uses the inverse relation *?isPrerequisiteOf(low\_concept, high\_concept)* as a *test goal* to verify the relationship between the given concepts, see Chapter 5, section 5.2. Thereafter to the *?hasContentX(a, a\_URL)* *test goal* (where *X* represents one of *Text* or *Video*) that ascertains the existence of a belief fact before the release of a learning material URL.

In the work of Klapiscak & Bordini (2009) every property or predicate relation between concepts in their FOL representation were not shared among the ontological statements. So the predicates were used in the unification of semantic literal tracking and mappings of atomic facts or literals in the ontology. But our approach to *ontology concept* matching or unification is quite different from this work. This is because the predicates are shared amongst many relations. That is, the predicates are related to several unary or binary literals, respectively. For example, the *desired\_Concept* predicate is in multiple concept relations, and in Prolog-like syntax are:

```
desired_Concept(delete)
desired_Concept(insert)
desired_Concept(select)
```

or the *hasPrerequisite* predicate in their binary relations

```
hasPrerequisite(delete, insert)
hasPrerequisite(insert, select)
```

that are similar to Gelfond (2008) and Zini & Sterling (1999) KB facts collection for a system. Thus, to ensure the right search and match of predicate statement in the collection of beliefs (i.e. the updated beliefs and initial beliefs representation) within a *hearer* agent BB, one of the literals, that is, either the *subject* or *object* as in *predicate(subject, object)* had their named-literal specified. For example

```
hasPrerequisite(X, insert)
```

which made ontological representation and communication more explicit for agents. This also facilitated the execution of the right plans, which includes the appropriate *achievement goals*, and other *actions* in the plan *body* as well as right replies to a *sender* agent where replies are required from the use of the *askOne* performatives. In contrast to the foregoing, it was realised that where two variables X and Y are given such as in

```
hasPrerequisite(X, Y)
```

binary relation, the *hearer* agents executed the wrong plan: because of the several relations in the ontology with the same predicate *hasPrerequisite* and same *subject* X.

Consider the following representation and its inter-agent communication. In a situation where both atomic literals are named in the relation

```
.send(agMaterial, askOne, hasPrerequisite(insert, delete))
```

the *hearer* agent (e.g. *agMaterial*) clearly distinguished the fact in its beliefs and made the appropriate and required *reply*. But the following message

```
.send(agMaterial, askOne, hasPrerequisite(X, Y)) . .(i)
```

gave room for ambiguity as the agent could not exactly map X to *insert* and Y to *delete* for instance, due to multiple representation with the same predicate *hasPrerequisite*. Thus, for the agent to unify its relational representations

appropriately during communication, the binary relation such as in (i) above was then structured to have at least a named-literal or concept such as in (ii) below:

```
.send(agMaterial, askOne, hasPrerequisite(X, delete)) . (ii)
```

where variable *X* is the desired concept of the student. The emphasis is that with at least one named literal in a binary relation, the actual fact needed to be unified were matched by the agent and the appropriate plan also selected for execution. The binary relations such as explained in (ii) was then adopted for all message communication to the agent *agMaterial*. For example, see the message with the *achieve* performative in (iii) below:

```
.send(agMaterial, achieve, hasPrerequisite(delete, Y))..(iii)
```

in which *Y* is an atomic variable that are instantiated by the agent easily without *confusion* about the appropriate plan. It is of importance to state that, on receipt of the message (ii), the *hearer* agent *agMaterial* initiates a *reply* message back to the *sender* agent. This reply, updated and created additional *fact* to the beliefs of the *sender* agent thus causing changes to the *sender* agent's *mental state*. The semantic operability of the *achieve* performative as given in message (iii) does not form a belief addition to the *hearer*'s beliefs.

Communication in a MAS can be *Assertive*, *Directive*, *commissive* or *Declarative* (Searle, 1969). The *achieve* performative is thus a *directive* (Searle, 1969) that gives a command to the *hearer* agent. At the message reception, the *hearer* agent adopts this performative message as a *goal* to execute—having got the plan to execute it.

Effective communication is bidirectional—between two entities that are either similar or dissimilar. In a MAS, communication is established when the message content of the *sender* is understood and utilised by the *hearer*, see Chapter 3. Some messages form belief addition, and some do not. This is dependent on the *performative* acts.

## 7.4 Agent *agInterface*: The Interface Agent

The process of communication in the MAS begins at the CArTAgO artifact when the agent *agInterface* observes percepts. A system that observes *percepts* or that takes inputs must have a *reactive* layer. The agent *agInterface* is the first reactive agent to the external world (of the user). In the process of fulfilling its functions within the MAS, the actions undertaken by the agent *agInterface* is described as both *Assertive* and *Directive* (Searle, 1969). The agent *agInterface* exercises its *Assertive* property, which is to inform, by observing and telling other agent in the MAS about the state of the environment—the *partially observable* environment (Wang, 2014): a non-natural environment since agents are not directly situated in the student. From *Assertive*, a *Declarative* act which is bringing changes by utterances is performed. This is actualised by belief change in the world (other agents) due to their belief updates from percept communication.

### 7.4.1 Percept Observation

Using the *Pre* and *Post* condition (Labrou & Finin, 1998), the task of observing by the agent *agInterface* is outlined as:

*Pre: value(V)[source(percept)] //environment percept*

*Post: send observed value(V) percept*

The *Pre* condition is the fact that must exist prior to the act of utterance. This is the percept obtained by the agent. The *value* predicate in the *value(V)* is the observation property configured in the CArTAgO environment (Ricci, Piunti & Viroli, 2011, see *Figure. 5.16*). The *Post* is the fact established after the act (utterance) is performed. This is an action performed in the plan body of the agent. Going by the nature of the pre-assessment MAS application that is meant to support teaching and learning, the use of the single predicate *value* as in

Value (V)

by the agent *agInterface* in the collection of percepts has been applied to all percepts. This includes the desired concepts and all SQL sentences (i.e. correct and incorrect answer queries) from students.

For example, consider the *DELETE* concept is the chosen *desired\_Concept* of a student that was submitted and perceived by the agent. From amongst the alternatives of *desired\_Concepts* (Fig.7.1 below) represented in FOL in the plan context (*i.e. precondition*), the **value("DELETE")** satisfied one of the specified conditions for the agent *agInterface* to adopt the plan. The adoption **+value("DELETE")** of this plan, triggers the execution of the plan *body* and the content is communicated to the named agent *agSupport*.

```
// agent agInterface perceive student's desired concept via percept

+value(V) [source(percept)] : value("SELECT") | value("INSERT") |
value("DELETE") | value("UPDATE") | value("JOIN") | value("UNION")
<-println("The topic you have entered to learn is: ", V);
.send(agSupport, tell, value(V));
.println("").
```

**Fig.7. 1:** List of desired SQL concepts contained in a plan context and a tell Performative as means of Communication.

This percept in the predicate `value(V)` is communicated to the agent *agSupport*—the pre-assessment agent, and received in its FOL logic form with the source as annotation

```
value("DELETE") [source(agInterface)].
```

In Figure 7.2 is the plan that receives students' SQL query answers. For students to learn SQL query construct professionally, assessment should be open-ended, not in multiple-choice alternatives. Thus the expected SQL answer queries to the System are open-ended. While the correct answers to SQL questions can be predetermined to compare with students' correct answer, the incorrect answers of students cannot be predetermined as there are bound to be varying answers from students to the same questions which signals a gap in learning. To gauge the level of skills and competencies, the queries expected in the system are made open-ended. But with one

*condition* that the values submitted to the system must not be empty by the use of a negation `not value("")`.

```
// agent agInterface perceive student's answer via percept

+value(V) [source(percept)] : not value("")
<-println("The answer you have provided is: ", V);
.println("");
.send(agSupport, tell, value(V));
.wait(600000).
```

**Fig.7. 2:** Plan for Perceiving the SQL Answer Queries from the student environment.

As agents communicate messages, their belief states are updated leading to experiential knowledge increase. From amongst the updated knowledge, a receiver agent becomes committed—a *commissive* act—to execute *intentions* which are contained in its plans. The plan which is executed is determined by the specified *context* in the plans.

## 7.5 Agent agSupport: The Pre-assessment Agent

This is the agent responsible for the *executive* functions of the pre-assessment process. The agent *agSupport* is the agent in the MAS that interrogates students' learning and the agent with most number of communications, see Chapter 4, Figure 4.7 for the *System Overview Diagram*.

At the observation of *value(V)* by the agent *agInterface*, if the content of the variable *V* that is communicated to the agent *agSupport* is a *desired\_Concept*; the variable *V* is substituted for the variable in the predicate statement *desired\_Concept(V)* in the agent plan (Fig. 7.3) that is contained in the `.send()` statement to the *agModel*, and *agModelling* to start the process of classification. After testing students, *agSupport* communicates the decision statement reached in every plan to the agent *agModelling* (*classifier*) that applies the principle of *learning by being told* to classify students. From Labrou & Finin (1998), the following are the *Pre* that describes the FOL data structure and the necessary beliefs that must hold before the agent *agSupport* proceeds with the *Post* conditions:

*Pre:* quizOfLeafnodes(X)[source(self)] //B

*Pre:* value(V)[source(sender)] //percept

*Post:* Adopt a desired\_Concept in the predicate value(V) [source(sender)]

*Post:* inter-communicate the desired\_Concept

*Post:* adopt an achievement goal in a plan to retrieve quiz from beliefs and display

*Post:* adopt a SQL query answer in the predicate value(V) [source(sender)]

*Post:* check whether SQL query answers in predicate value(V)[source(sender)]

*Post:* [passed or failed] decision

*Post:* send a passed or failed predicate message

The representation

**Pre:** quizOfLeafnodes (X) [source (self) ]

that is annotated with [source(self)] (see Chapter 3, section 3.12.1) are a collection of initial knowledge of questions from which students are pre-assessed by the agent. Jason agent knowledge can be of source (self) , source (percept) , or source (sender) (Bordini, Hubner & Wooldridge, 2007). The **Post** are the actions undertaken by the agent as given. Aside these, some other **Post** condition actions are the concatenation of *date* i.e. date (YY, MM, DD) and *time* i.e. time (HH, NN, SS) functions to all the percepts received before their communication to other agents. The .concat () is a Jason internal action that joins strings in a specified variable.

### 7.5.1 The Agent Pre-assessment Process

The agent *agSupport* receives the concept *value("DELETE")*[source(agInterface)] communicated by source: [source(agInterface)]. The agent *agSupport* has been initialised with the beliefs of prerequisite questions as knowledge in unary predicate, as shown with the **Pre** condition, from where it can fetch or instantiate the required facts during pre-assessments sessions. Based on the current knowledge state of the agent e.g. +*value("DELETE")* percept, the perceived communicative message triggers the *plan* to display the prerequisite questions when the pre-condition is matched. The



Figure 7.3 depicts this process including other detailed communication protocol and *date and time stamping* of users' activities through to the *!achievement goal* (or desire *w.r.t.* BDI)

```
!quizInsertSelect(InsertSelectQuiz) //D
```

that the agent wants to realise with a variable `InsertSelectQuiz` that is matched with the unary representation in the agent beliefs when the agent adopts this goal (e.g. Fig. 7.6).

```
@plan15_Delete_desiredConcept
+value(V) [source(agInterface)] : value(V) == value("DELETE")
<-.date(YY, MM, DD); .time(HH, NN, SS);
  .send(agModelling, tell, desired_Concept(V));
  .send(student, tell, desired_Concept(V));
  .concat(V, ", date(", YY, "-", MM, "-", DD, ")", ", ", "time(", HH, "-",
  NN, "-", SS, ")", MsgD);
  .send(student, tell, desired_Concept(MsgD)); //date and time appended
  .send(agMaterial, askOne, hasPrerequisite(V, insert)); //Asking if
  relation exists in ontology
  .println(V, " has prerequisite INSERT");
-value(V); //belief drop
.println("Question on INSERT SELECT:");
!quizInsertSelect(InsertSelectQuiz);
.println.
```

**Fig.7. 3:** Adoption of the DELETE desired Concept.

As the variable name `InsertSelectQuiz` indicates, the first leaf-node question corresponding to the `InsertSelect` of the immediate prerequisite class to `Delete` is released, *see Figure 5.3*. The unit of lessons or learning are the leafnodes that contains the SQL queries. Hence, the programming of *!achievement goals* of the agent *agSupport* to the leafnodes of the SQL ontology structure.

On receipt of the SQL query answer i.e. percept to the first prerequisite question from the agent *agInterface*, the agent *agSupport* selects the relevant plan to assess the student's SQL query skill using the *passed* or *failed* boolean predicate states given in the agent respective plans. For a given leafnode, each plan compares all SQL query answers. While the plan for the *passed* predicate decision compares student correct

answer with the use of equality == operator; the plan for the *failed* predicate decision compares the incorrect SQL answer using the *different* \== Prolog operator, (as described in Chapter 3). This type of comparison operators also applies to Jason AOP. With the \== operator, the agent returns *true* for all its perceived inputs. The implication of this is that the agent was unable to navigate or move from one incorrect SQL answer plan to another. Now to aid the agent navigation from plan to plan selection and execution, Jason FOL iterative statements were introduced as part of the constraints in the agent plan *context*. The Figure 7.4 and Figure 7.5 code snippets are two examples of plans: one each for a *correct* and *incorrect* SQL query answer, respectively; with respect to the `Insert_Select`. Notice the *!achievement goal*

```
!quizInsertValue(InsertValueQuiz) //D
```

at the end of the plans in the Figures 7.4 and 7.5.

```
@plan14_InsertSelect_correct
// Plan for correct answer to INSERT with SELECT of the DELETE desired_Concept.

+value(V) [source(agInterface)] : value(V) == value("INSERT INTO
TENNIS_RECR_PLAYERS (PLAYERNO, NAME, TOWN, PHONENO) SELECT PLAYERNO, NAME, TOWN,
PHONENO FROM TENNIS_PLAYERS WHERE LEAGUENO IS NULL") & countForDeletePre(0)
<- .date(YY, MM, DD); .time(HH, NN, SS);
.println("Good. Your answer is correct.");
?countForDeletePre(Count); -+countForDeletePre(Count+ 1);
.concat(V, ", date(",YY,"-", MM,"-", DD, ")", " ", " ", "time(",HH, "-", NN, "-",
SS, ")", Ris);
.send(student, tell, responseToInsertSelect(Ris)); //date and time appended
PassedIS = "The student has passed the INSERT with SELECT question.";
.concat(PassedIS, ", date(",YY,"-", MM,"-", DD, ")", " ", " ", "time(",HH, "-", NN,
"-", SS, ")", Pis);
.send(student, tell, passed(Pis));
.send(agModelling, tell, passed(PassedIS));
.println("Next question on INSERT VALUE:"); .println;
!quizInsertValue(InsertValueQuiz); .println.
```

**Fig.7. 4:** Plan for a Passed Pre-assessment of InsertSelect

```

@plan12_InsertSelect_incorrect
// INSERT with SELECT question of the DELETE desired_Concept.

+value(V)[source(agInterface)] : value(V) \== value("INSERT INTO
TENNIS_RECR_PLAYERS (PLAYERNO, NAME, TOWN, PHONENO) SELECT PLAYERNO, NAME, TOWN,
PHONENO FROM TENNIS_PLAYERS WHERE LEAGUENO IS NULL") & countForDeletePre(0) & not
value("UNION")
& not value("JOIN") & not value("SELECT") & not value("INSERT")
<- .date(YY, MM, DD); .time(HH, NN, SS);
  ?countForDeletePre(Count); +-countForDeletePre(Count+ 1);
  .concat(V, ", date(",YY,"-", MM,"-", DD, ")", ", ", "time(",HH, "-", NN, "-", SS,
  ")", Ris);
  .send(student, tell, responseToInsertSelect(Ris)); //date and timestamp
  .println("You have NOT passed the INSERT with SELECT question.");
  FailedIS = "The student has NOT passed the INSERT with SELECT question.";
  .concat(FailedIS, ", date(",YY,"-", MM,"-", DD, ")", ", ", "time(",HH, "-", NN,
  "-", SS, ")", Fis);
  .send(student, tell, failed(Fis));
  .send(agModelling, tell, failed(FailedIS));
  .println("NEXT Question on INSERT VALUE:");
  !quizInsertValue(InsertValueQuiz).

```

**Fig.7. 5:** Plan for a Failed Pre-assessment of InsertSelect, and giving agent the subgoal !quizInsertValue(InsertValueQuiz)

This is the agent sub-goal to be realised and it represents the next prerequisite question on the InsertValue (the second leafnode and neighbour to the InsertSelect). When *achievement goals* are adopted e.g. +!quizInsertValue(InsertValue), questions are presented to students. The Figure 7.6 shows the adoption of the *achievement goal* that actualises the release of the InsertValue question. As visibly shown in Figure 7.6, the pre-condition in the agent plan *context* is a necessary condition that must exist in its beliefs for the agent to decide or be committed this *intention* w.r.t. BDI (see Bordini, Hubner & Wooldridge, 2007).

```

+!quizInsertValue(InsertValueQuiz) : quizInsertValue(InsertValueQuiz)
<- .date(YY, MM, DD); .time(HH, NN, SS);
  .println("Question on INSERT VALUE:");
  .println(InsertValueQuiz);
  .concat(InsertValueQuiz, ", date(",YY,"-", MM,"-", DD, ")", ", ",
  "time(",HH, "-", NN, "-", SS, ")", Qiv);
  .send(student, tell, quizInsertValue(Qiv));
  .wait(6000000).

```

**Fig.7. 6:** Adoption of +!quizInsertValue achievement goal, display and communication.

The number of plans for pre-assessment in the agent *agSupport* has been determined by the number of leafnodes considered under a given *desired\_Concept* such that every leafnode has two pre-assessment plans: one for a *passed* pre-assessment and other for a *failed* pre-assessment. In the DL ontology (Chapter 5), the number of leafnode per parent class has been defined to have *leafnode*  $N \geq 2$  *minimum cardinality*.

Also note that in the Figure 7.3 that, the *.send()* *internal action* has the *tell* and *askOne* performatives. These performatives have been used by the agent *agSupport* to communicate knowledge and to make enquiries, respectively. The *tell* sends messages e.g. a student *desired\_Concept*, correct, and incorrect answers; to other agents such as the agent *agModel* (*student*, or *TextPersistent* agent). However the *askOne* in

```
.send(agMaterial, askOne, hasPrerequisite(V, insert));
```

is a message that requests the receiver agent *agMaterial* whether the variable *V* unified with a literal in the statement *hasPrerequisite(V, insert)* in the agent's ontological beliefs.. This is a communication that does not add beliefs to the receiver agent *agMaterial*, but makes the agent *agMaterial* reply to the content that matched the binary representation. The reply to the agent *agSupport* caused belief addition, and in turn was used by the agent to display the information to the student user that

```
delete hasPrerequisite insert
```

where *insert* is the prerequisite to be pre-assessed.

Jason is an extension of the AgentSpeak language which is BDI programming language (Bordini, Hubner & Wooldridge, 2007; Bădică *et al.* 2011). As noticed in the plan *context* of Figures 7.5 for example, the use of constraints for controlling the selection of plans in agent programs is not uncommon. Padgham & Singh (2013) state that to make sure that a preferred plan is selected by an agent, most BDI programs are often filled with constraints that narrows down the selection of a plan. This accounts for the number of constraints in the agents *agSupport* and *agModelling* in this thesis. As stated earlier in Chapter 3, plans are a list of courses of action that are executed in turns. In the Pre-assessment System, just as one agent *plan* triggers another agent *plan* through inter-message communication, so, within the agent *agSupport*, one *plan* has triggered another plan through the use of *achievement goals* adoption. This is done until the agent navigates through the questions corresponding to all the leafnodes considered under a given desired concept, (*as first described in Figures 4.21 and 4.22*).

## 7.6 Strategies of the Pre-assessment System Development

As earlier mentioned, leafnodes are the concepts which students are pre-assessed on, not the parent class concepts. Pre-assessment on a leafnode is either a *passed* or *failed* outcome; such as in

```

IF (answer is correct)
    THEN (actions for correct answer)
        !achievement goal
IF (answer is incorrect)
    THEN (actions for incorrect answer)
        !achievement goal

```

where the *!achievement goal* of the pair of the correct and incorrect answers to a given leafnode is towards this same leafnode.

### 7.6.1 Pre-assessment By Immediate Prerequisite Class Program Development

Given the regular ontology (Figure 5.3), in the agent *agSupport* program, there are two pre-assessment plans per leafnode, and one agent plan each per *desired\_Concept* that

begins the pre-assessment process. Then it means that, in the ontology, each parent class and its two leafnodes has a sub-total of 5 plans. In the ontology, the Union class concept has no super class. Therefore, as shown in the agent plans, pre-assessment begins with the leafnodes of the immediate lower class i.e. the Join. Underneath the Union class, there are 5 parent classes which are the Join, Update, Delete, Insert and Select where each parent class and their leafnodes have 5 plans, respectively. Therefore, the total number of pre-assessment plans in the agent *agSupport* amounts to  $25 + 1 = 26$  plans, where 1 is the plan that represents the lowest class concept that has no prerequisite as symbolised with the letter A in the *Pre-Assessment Mechanism*, Figure 4.18. This excludes any plan for the leafnodes UnionAll and UnionDistinct because the parent class has no superclass.

- **Iterative Control Statement**

This section describes the iteration that has been used to enable the agent *agSupport* to navigate between its own plans. This began by first initialising the iteration statement to zero in the agent *agSupport* beliefs i.e. `testCount(0)` (Fig. 7.7). For the ontology of equal leafnodes, the same predicate (also known as *functor*) “`testCount()`” was applied to all iterative control statements in agent *agSupport* pre-assessment plans in the strategy of *Pre-assessment By Immediate Prerequisite Class*.

```
//agent agSupport in project preassessment.mas2j

/* initial belief and facts */
testCount(0).
```

**Fig.7. 7:** Initialising an iteration belief.

```
?testCount(Count);
--testCount(Count + 1);
```

**Fig.7. 8:** Testing and updating the iteration in a plan body.

Because the ontology being considered is a regular ontology of two leafnodes per parent class, the iteration is also equal to 2, with 1 iteration e.g. `testCount(1)` being shared by the plans of both a correct SQL query answer and incorrect SQL query answer that corresponds to a leafnode concept. In that light, the execution of the iteration is thus dependent on either of the answers that is entered by a student. Recall that the number of leafnodes determines the Boolean parameter [P or F] combinations and number of classification rules, see Chapter 4, section 4.8 and 4.9. Thus, based on a regular ontology of 2 leafnodes, a total of four possible classification categories per parent class was drawn for the agent *agModelling*. On the receipt of answer percept and execution of a plan by the agent *agSupport*, the Jason iterative statement is updated as shown in Figure 7.8. The decision tree in Figure 7.9 diagrammatically presents how students are classified into one of the following categories: <PP>, <PF>, <FP> or <FF> given, for instance, the DELETE concept.

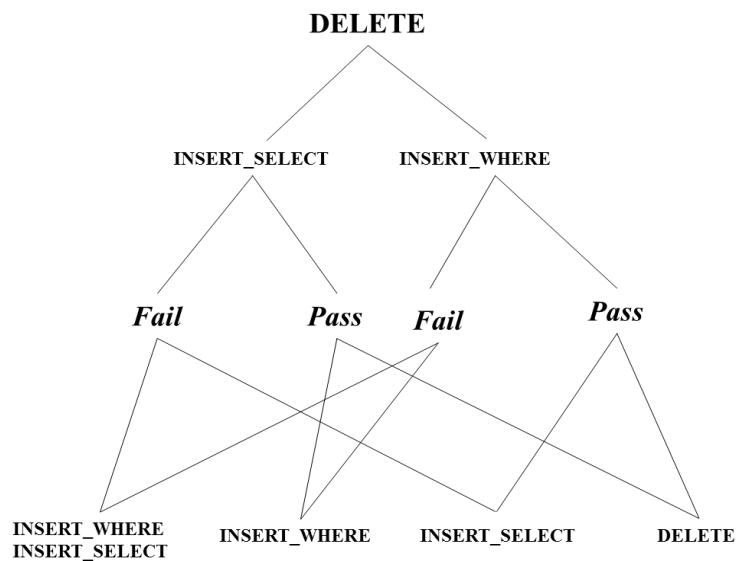


Fig.7. 9: Classified Decision Tree Flow for DELETE Pre-assessment

### 7.6.2 Pre-assessment By Multiple Prerequisite Classes Program Development

The strategy of *Pre-assessment By Multiple Prerequisite Classes* is that in which additional leafnodes of two more prerequisite classes underneath a given *desired\_Concept* is considered for pre-assessment. This strategy involves the

navigation of agent plans and its *achievement goals* from one plan to another in the order of SQL learning concepts in Figure 5.1 and also across multiple classes. To demonstrate the multiple prerequisite assessment and classification process, the Figure 5.4 which is a non-regular ontology has been considered for the application of this strategy. The ontology model is *non-regular* because the number of leafnodes across its parent class nodes are not equal in number. To be precise, the `Select` class node has leafnodes  $N = 4$  as against `Join` that has  $N = 3$ , and others have  $N = 2$ . The TABLE 7.1 presents an order of multiple class pre-assessment from a given desired class concept, through its prerequisites classes, down to all leafnodes  $N$ .

**TABLE 7. 1: DESIRED\_CONCEPT AND ORDER OF MULTIPLE PREREQUISITES CLASS FOR PRE-ASSESSMENTS BASED ON FIGURE 5.4**

Desired_Concept	Prerequisite classes	Prerequisite leafnodes	No. of leafnodes N
<i>Select</i>	<i>No prerequisite</i>	<i>Nil</i>	<i>Nil</i>
<i>Insert</i>	$\exists$ hasPrerequisite.{select}	<i>selectOrderBy,</i> <i>selectDistinct,</i> <i>selectWhere,</i> <i>selectAll</i>	4
<i>Delete</i>	$\exists$ hasPrerequisite.{insert, select}	<i>insertSelect,</i> <i>insertValue,</i> <i>selectOrderBy,</i> <i>selectDistinct,</i> <i>selectWhere,</i> <i>selectAll</i>	6
<i>Update</i>	$\exists$ hasPrerequisite.{delete, insert, select}	<i>deleteSelect,</i> <i>deleteWhere,</i> <i>insertSelect,</i> <i>insertValue,</i> <i>selectOrderBy,</i> <i>selectDistinct,</i> <i>selectWhere,</i> <i>selectAll</i>	8
<i>Join</i>	$\exists$ hasPrerequisite.{update, delete, insert, select}	<i>updateSelect,</i> <i>updateWhere,</i> <i>deleteSelect,</i> <i>deleteWhere,</i>	



		<i>insertSelect,</i> <i>insertValue,</i> <i>selectOrderBy,</i> <i>selectDistinct,</i> <i>selectWhere,</i> <i>selectAll</i>	10
<i>Union</i>	$\exists$ hasPrerequisite.{ <i>join,</i> <i>update, insert, select</i> }	<i>selfJoin,</i> <i>fullOuterJoin,</i> <i>innerJoin,</i> <i>updateSelect,</i> <i>updateWhere,</i> <i>deleteSelect,</i> <i>deleteWhere,</i> <i>insertSelect,</i> <i>insertValue,</i> <i>selectOrderBy,</i> <i>selectDistinct,</i> <i>selectWhere,</i> <i>selectAll</i>	12
<b>Desired_Concept</b>	<b>Prerequisite classes</b>	<b>Prerequisite leafnodes</b>	<b>No. of leafnodes N</b>

For example, on the *Union desired\_Concept* with the leafnodes (or units of lessons) as the *UnionAll* and *UnionDistinct* that a student intends to learn; the student would need to be pre-assessed on all prerequisite leafnodes underneath the *Union* as shown in the TABLE 7.1. This type of arrangement is at variance with the educational principle of *Chunking* (Casteel, 1988; Anderson, 2008) in which the presentation of classified learning materials is prescribed in “smaller quantities” for students to succeed. This theory is required in the design of a formative assessment system for SQL: a subject area that has been adjudged as challenging and difficult to learn (Mitrovic, 1998; Prior, 2003). Thus the Pre-assessment MAS is a formative assessment system that has engaged the principle of *Chunking* in its design to facilitate effective learning in students. Based on the background literature on the difficulty experienced by students in SQL, and the results obtained so far from the Pre-

assessment System evaluation, managing this units of learning in smaller quantities would enable students to be more successful in their learning of SQL.

To demonstrate the strategy of *Pre-assessment By Multiple Prerequisite across* parent classes with respect to the *Chunking* educational principle of learning, the Figure 5.4 was remodelled into Figure 7.10. The following *illustration 1, 2, and 3* presents this strategy over a non-regular ontology in the pre-assessment process. In the Figures 7.10 - 7.13, the red arrows indicate the link between two classes, and the black arrows, the link between a class and its subclasses.

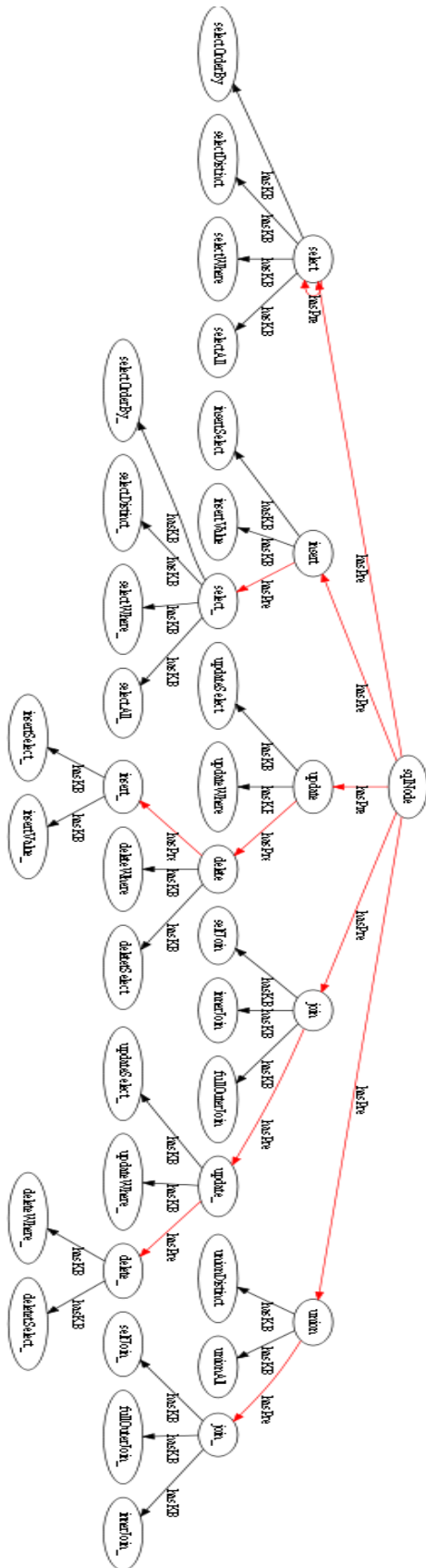
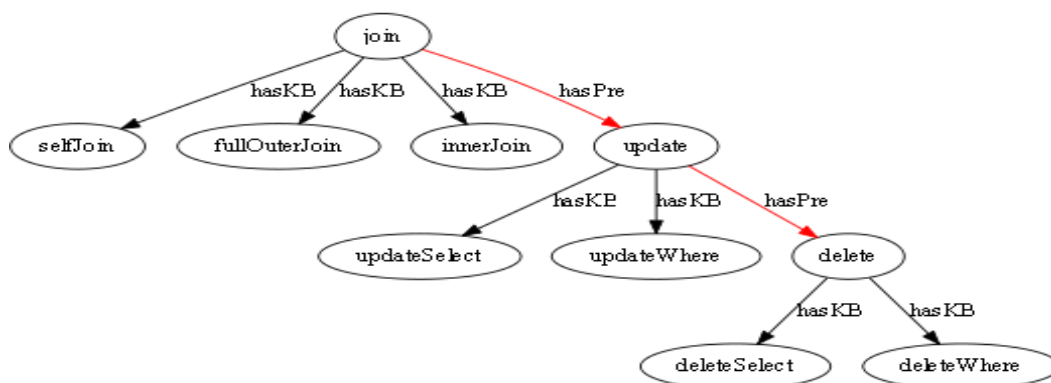


Fig. 7.10: A non-regular ontology

▪ **Illustration 1**

As already mentioned, Jason AOP is language that uses Prolog-like syntax. Prolog is a FOL language for demystifying complex DL formula (Almendros-Jemenez, 2011) for separating assertions from DL defined concepts. From the  $R(b, a)$  or  $p(a, b)$  binary expression, the `Join` concept and its relationships with other classes (Fig. 7.11) considered for multiple prerequisites are stated in FOL to produce some initial belief (see TABLE 7.2) for the pre-assessment MAS.

In the Figure 7.11, `Join` is a main topic (that represents a *desired\_Concept*) with `SelfJoin`, `FullOuterJoin` and `InnerJoin` as its unit of lessons (i.e. the leafnodes). Under `Join` are multiple *prerequisite* parent classes  $C$  that comprises the `Update` and `Delete` concepts, both with a total number of leafnode  $N = 4$ ; namely: `UpdateSelect`, `UpdateWhere`, `DeleteSelect` and `DeleteWhere`. In the TABLE 7.2 we show the relationship between these classes and their leafnode concepts. In the TABLE are agent initial beliefs of the named concepts (as represented in the system), agent *achievement goals* and the pre-assessment process for the `Join` learning target. The *achievement goals* e.g. `!quizUpdateSelect` are the goals given to the agent to quiz a student. From plan to plan, they serve as links that connects the ontological nodes in a tree for pre-assessments. Like in Prolog programs, navigation between plans in Jason ends with full stops “.”, which implies the logical OR between plans. Also, inside agent plans are statements that breaks with semi colon “;” that implies the logical AND.



**Fig.7. 11:** Semantic relations of a total of 4 prerequisite leafnode of two prerequisites parent classes under `Join`.

TABLE 7. 2: THE JOIN PRE-ASSESSMENT PROCESS ILLUSTRATION

Initial ontology belief state	Pre-assessment process: IF...THEN	Agent achievement goal
hasPre(join, update)	<b>IF</b> Join	
hasKB(update, pdateSelect)	<b>THEN</b> updateSelect	!quizUpdateSelect
hasKB(update, pdateWhere)	updateWhere	!quizUpdateWhere
hasPre(update, delete)		
hasKB(delete, deleteSelect)	deleteSelect	!quizDeleteSelect
hasKB(delete, deleteWhere)	deleteWhere	!quizDeleteWhere

▪ **Illustration 2**

In the Figure 7.12 is the *desired\_Concept* Insert with InsertSelect and InsertValue as its unit of lessons (leafnodes). But under Insert is one *prerequisite* Select with leafnodes N = 4, namely: SelectOrderBy, SelectDistinct, SelectWhere, and SelectAll that also represents the Select unit of lessons. In TABLE 7.3 are agent initial beliefs, agent *achievement goals* and the pre-assessment process when the Insert is a student’s target of learning.

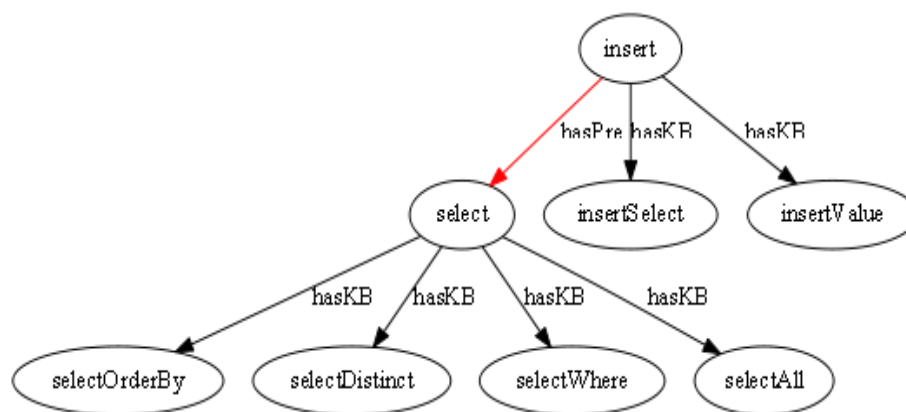


Fig.7. 12: Semantic relations of a total of 4 prerequisites leafnode for pre-assessment under the Insert.

TABLE 7. 3: THE INSERT PRE-ASSESSMENT PROCESS ILLUSTRATION

Initial ontology belief state	Pre-assessment process: IF...THEN	Agent achievement goal
hasPre(insert, select)	<b>IF</b> insert	
hasKB(select, selectOrderBy)	<b>THEN</b> selectOrderBy	!quizSelectOrderBy
hasKB(select, selectDistinct)	selectDistinct	!quizSelectDistinct
hasPre(select, selectWhere)	selectWhere	!quizSelectWhere
hasKB(select, selectAll)	selectAll	!quizSelectAll

▪ **Illustration 3**

Pre-assessment based on UNION as *desired\_Concept* in which its unit of lessons (leafnodes) are UnionAll and UnionDistinct is over the instances of the Join prerequisite that has prerequisite leafnode  $N = 3$ , namely: SelfJoin, FullOuterJoin and InnerJoin (Fig. 7.13). TABLE 7.4 also illustrates the relations between these unit of lessons and the process of agent goal achievement.

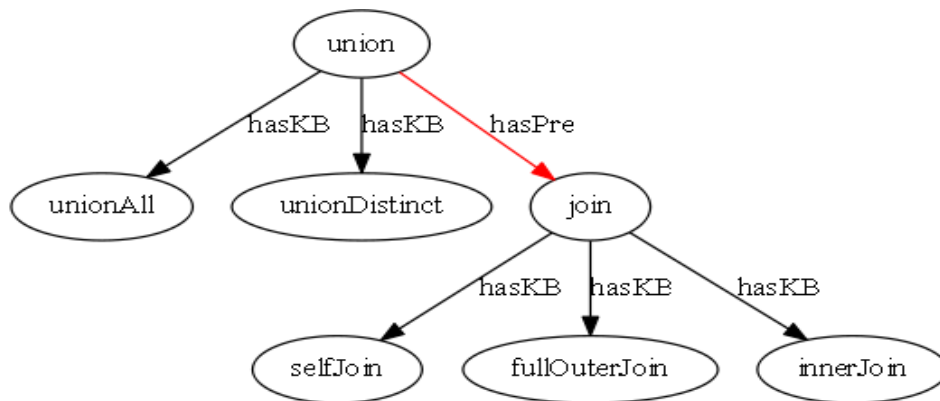


Fig.7. 13: Semantic relations of a 3 prerequisite leafnodes under the Union *desired\_Concept*.

TABLE 7. 4: THE UNION PRE-ASSESSMENT PROCESS ILLUSTRATION

Initial ontology belief state	Pre-assessment process: IF...THEN	Agent achievement goal
hasPre(union, join)	<b>IF</b> union	
hasPre(join, selfJoin)	<b>THEN</b> selfJoin	!quizSelfJoin
hasKB(join, fullOuterJoin)	fullOuterJoin	!quizFullOuterJoin
hasKB(join, innerJoin)	InnerJoin	!quizInnerJoin

By analogy the arrangement of plans for both the *Pre-assessment By Multiple Prerequisite Classes* strategy and that of the *Pre-assessment By Immediate Prerequisite Class* strategy in the agent *agSupport* follows the same procedure. This is shown in the *pseudo-algorithm* in Figure 7. 14. The *Multiple Prerequisite Classes* strategy involves the process of given agent *!achievementn goals* to navigate more plans to cover additional prerequisite leafnodes as shown in the *Illustrations 1, 2 & 3* based on Figure 7.10 non-regular ontology. As a result of the variation in the leafnodes, plans for the respective *desired\_Concept* class were programmed to use a different *functor* in their iterative statement: one per parent class, where

- Each iterative statement is initialised to 0, and begins at the first plan that corresponds the (*passed or failed*) answers of first leafnode prerequisite to the *desired\_Concept*;
- A correct and incorrect plan equally shared one iteration; and
- The iterations as constraints in a plan *content* and pre-conditions.

The iterative statements in Figure 7.15 are the initialised iterations for the answers of the prerequisite plan for the Union, Join, Update, Delete, and Insert *desired\_Concepts*, respectively. The iterations are aids for the agent to navigate down its plan. This was introduced during development, because the agent would continuously execute only the first plan of the plans corresponding to the incorrect SQL query answers. This approach provided a solution.

---

**Pseudocode of pre-assessment and interaction in the multiagent system**


---

```

1. initial beliefs: predicate(Class, Class)
2. initial beliefs: predicate(Class, Leafnode)
3. initial beliefs: predicate(Leafnode, URL)
4. initial beliefs: quiz(PrerequisiteLeafnode)
5. Given a desired concept that has N leafnodes prerequisite
6. IF
7.   Percept ← desiredConcept
8. THEN
9.   .send(receiver, tell, desiredConcept)
10.  fetch the next quiz(Prerequisite_Leafnode)
11.  .send(receiver, tell, quiz(Prerequisite_Leafnode))
12.  output quiz(Prerequisite_Leafnode)
13.  Percept ← answer(X)
14.  IF
15.    answer(X) == answer(Prerequisite_Leafnode)
16.  THEN
17.    passed(Prerequisite_Leafnode) decision
18.    .send(receiver, tell, passed(Prerequisite_Leafnode))
19.  IF
20.    answer(X) \== answer(Prerequisite_Leafnode)
21.  THEN
22.    failed(Prerequisite_Leafnode) decision
23.    .send(receiver, tell, failed(Prerequisite_Leafnode))
24. IF
25.   N number of leafnodes have been pre-assessed on
26. THEN
27.   .send(receiver, achieve, recommendMaterial)
28. Else
29.   repeat 10 to 27

```

---

**Fig.7. 14:** Pseudo-algorithm of the pre-assessment process that depends on the number of leafnodes N considered under a *desired\_Concept*

```

/* initial belief and facts */
countForInsertPre(0).
countForDeletePre(0).
countForUpdatePre(0).
countForJoinPre(0).
countForUnionPre(0).

```

**Fig.7. 15:** Initialisation of iterations as beliefs in agent *agSupport*



### 7.6.3 Open\_Ended Answers Assessment

Programming a MAS for the recognition of *negative facts* (i.e. incorrect answers) can pose some difficulty for agent plan selection and execution of goals when the expected inputs are limitless in scope, unbounded or open-ended texts. It is quite different when it is of *positive facts* i.e. the correct answers.

With positive facts, the expected input answers were represented in the agent such that when the perceived percept was matched in a plan, relevant plans were selected and actions in the *body* of the *plan* executed. This is because positive facts are information whose representation are known and can be represented or given to agent for comparison with incoming percepts. But negative facts are unknown and as such cannot be pre-determined for representation, yet database student needs to program SQL like professionals (Prior, 2003). In order for database students (in this study) to program like professionals, they needed to code their *resultset* queries on the Pre-assessment System. This was aimed at revealing their line of thoughts and unravelling the technical difficulty faced in SQL by pointing them to relevant materials, and to better inform teaching strategy.

So, with the *open-ended* nature of SQL queries, comparisons of perceived incorrect SQL answer inputs are assessed with the *different* `\==` operator. But this was without inconsistency in the agent behaviour at the time of System development. This was when an answer input does not match the positive fact or correct answer. The `\==` operator caused previous or existing beliefs to trigger irrelevant plans. To enable the agent *agSupport* to select the plans that uses the `\==` operator, iterative statements such as `countForDeletePre(X)` (Fig. 7.15) were introduced in the agent plan *context*. This was also coupled with some negated predicate statement such as `not value("INSERT")` to block existing or incoming percept from soliciting un-required plans.

## 7.7 Agent agModelling: The Task of Classification

Classification in this thesis is the technique used in categorising students' skill status in order to recommend learning materials that meets their learning needs. The task of classification is that of the agent *agModelling* (the *classifier*). The process of

classification starts with the inter-agent communication of the *desired\_Concept* of students to this agent, as shown in the *Prometheus PDT* diagrams in Chapter 4, e.g. *Figure 4.9*, and *Figure 4.14*. This marks the beginning of the search for a student's class for material recommendation that ends with the last of the prerequisite leafnodes under a *desired\_Concept*.

To classify, the agent combines a set of predicate statements such as *desired\_Concept(X)*, *passed(N)* and *failed(N)* to make a decision for the right level of skill. The process of rule formation which was described in Chapter 4 with the use of FOL syntax is the conjunction of the *desired\_Concept(X)*, *passed(N)* and *failed(N)* predicate decision messages received by this agent, where *N* in the FOL formulas *passed(N)* and *failed(N)* as in

```
passed("The student has passed the UPDATE with SELECT question")
and
failed("The student has NOT passed the DELETE with WHERE
question")
```

are not of the same leafnode in the same agent plan. These messages which are updated beliefs are the premise in which the classifier agent matches its plan *contexts* as well as adopts its *triggering\_event* before proceeding to execute the actions in the plan. Engaging the use of the **Pre** and **Post** conditions, the task of classifying is stated as:

```
Pre: desired_Concept(X) [source(sender)] //percept
Pre: passed(N)[source(sender)] //percept
Pre: failed(N) [source(sender)] //percept
Post: Adopt a plan where all Pre are satisfied, and classify
Post: send an achieve performative message
```

During the Pre-assessment System evaluation and participants skills' test sessions, students' SQL pre-skills status to a *desired\_Concept* were evaluated, classified, and appropriate recommendations made. When a plan *context* amongst its list of plans is satisfied, all that is contained in the plan *body* are actions of messages conveyed by the *achieve* performative. These actions are executed through the `.send()` *internal*

action to the agent *agMaterial* for the release of learning material URL(s). One `.send()` internal action with *hasKB* predicate represents one material recommendation, while that of *hasPrerequisite* predicate contains a collection of all the leafnodes of a *desired\_Concept*, or that of all the *failed* leafnodes of a prerequisite, see Figures 7.16 and 7.17 respectively. Thus, from logic based semantics, for a 4 leafnodes *N* underneath a *desired\_Concept*, the classification rule for the Fig. 7.16 can be explicitly stated as

$$\begin{aligned} & \forall \text{desiredConcept}(C) \forall N_4 \forall N_5 \forall N_6 \forall N_7 \\ & : \exists \text{desiredConcept}(C) \wedge \exists \text{passed}(N_4) \wedge \exists \text{passed}(N_5) \wedge \exists \text{passed}(N_6) \wedge \exists \text{passed}(N_7) \\ & \Rightarrow \text{desiredConcept}(C). \{N_1, N_2, N_3\} \end{aligned}$$

where the conclusion *N<sub>1</sub>*, *N<sub>2</sub>*, and *N<sub>3</sub>* are the prescribed leafnodes of the *desired\_Concept* that is recommended for learning for all the *passed* prerequisite leafnodes *passed(N<sub>4</sub>)*, *passed(N<sub>5</sub>)*, *passed(N<sub>6</sub>)* and *passed(N<sub>7</sub>)* in the *context* or condition part of the rule.

```

/* A classification rule for pre-assessments under the JOIN concept */
@joinRule1
+!recommendMaterial[source(agSupport)] : desired_Concept("JOIN")[source(agSupport)]
    & passed("The student has passed the UPDATE with SELECT question.")
    & passed("The student has passed the UPDATE with WHERE question.")
    & passed("The student has passed the DELETE with SELECT question.")
    & passed("The student has passed the DELETE with WHERE question.")
<- .send(agMaterial, achieve, hasPrerequisite(join, update)).

```

**Fig.7. 16:** Two multiple prerequisite classes of 4 leafnodes classification. Agent *agModelling* sending *hasPrerequisite* predicate message.

Similarly, for the Fig. 7.17, the applied logic based classification syntax is

$$\begin{aligned} & \forall \text{desiredConcept}(C) \forall N_4 \forall N_5 \forall N_6 \forall N_7 \\ & : \exists \text{desiredConcept}(C) \wedge \exists \text{passed}(N_4) \wedge \exists \text{failed}(N_5) \wedge \exists \text{failed}(N_6) \wedge \exists \text{failed}(N_7) \\ & \Rightarrow \text{failed}(N_5 \wedge N_6 \wedge N_7) \end{aligned}$$

where  $N_5$ ,  $N_6$ ,  $N_7$  are the prescribed and recommended leafnodes of the *failed* prerequisites, namely,  $failed(N_5)$ ,  $failed(N_6)$  and  $failed(N_7)$  in the *context* or condition part of the rule. Given that *context* is any information that can be used to characterise the situation of an entity: where an entity is a person, place or object (Dey, Abowd & Salber. 2001; Verbert *et al.* 2012). The stated axioms as implemented are the *modelled learning paths* (Bañeres, 2017) for individual students for a given *desired\_Concept*.

```

/* A classification rule for pre-assessments under the JOIN concept */
@joinRule4
+!recommendMaterial[source(agSupport)] : desired_Concept("JOIN")[source(agSupport)]
    & passed("The student has passed the UPDATE with SELECT question.")
    & failed("The student has NOT passed the UPDATE with WHERE question.")
    & failed("The student has NOT passed the DELETE with SELECT question.")
    & failed("The student has NOT passed the DELETE with WHERE question.")
<- .send(agMaterial, achieve, has_KB(X, update_where));
    .send(agMaterial, achieve, has_KB(X, delete_select));
    .send(agMaterial, achieve, has_KB(X, delete_where)).

```

**Fig.7. 17:** Two multiple prerequisite classes of 4 leafnodes classification. Agent *agModelling* sending *hasKB* predicate message.

During pre-assessment, the number of `.send()` *internal action* that is communicated to the agent *agMaterial* is determined by the performance of the student. But the number of classification rules and the parameters *passed* and *failed* combinations are determined by the number of leafnodes under a given *desired\_Concept* programmed at design time. The content of the `.send()` message of this agent *agModelling* are binary relation e.g.

```
.send(agModelling, achieve, has_KB(X, select_orderby))
```

in their FOL representations. These `.send()` *internal action* messages ranges from 1 to 4 *action* according to the strategies of the *Pre-Assessment By Immediate Prerequisite Class* and the *Pre-Assessment By Multiple Prerequisite Classes* explained earlier. At the end of pre-assessment, the classifier agent classifies students into one of the classified categories, namely:

- The *desired\_Concept* when all prerequisites are *passed* correctly,

- The *failed* leaf-node when some prerequisite is answered incorrectly, or
- All of the *failed* leaf-nodes when all prerequisites are answered incorrectly

with-respect-to the number of leaf-node  $N$  considered under a preferred *desired\_Concept*.

### 7.7.1 Generating Parameter Combination for Classification

Each leafnode  $N_{i,j}$  has two possible boolean states [*passed* or *failed*] upon which a student is pre-assessed. For a large number of leafnodes, say leafnode  $N \geq 4$  under a *desired\_Concept*, the process of estimating the required number of classification rules  $R$  has been given in Chapter 4. But the process of generating the rules via parameters [*passed* or *failed*] combinations for accurate classification for a number of leafnode  $N$  can also be tedious to derive, *see the FOL notation in Chapter 4*. Thus to combine the [*passed* or *failed*] parameters for accurate classification with respect to leafnodes  $N_x$ , the **Figure 7.18** presents the algorithm for the *classifier* agent.

---

#### Algorithm for Generating Classification Rules

---

1. Initialise  $T = [P, F]$  /\*\* pass or fail boolean parameter \*/
  2.  $1 \leq x \leq k$
  3. While  $x \neq k$
  4.      $N \leftarrow N_1, \dots, N_{x+1}$  /\*\* number of leafnodes \*/
  5.      $Initial\_Rule = T * (N_x)$  /\*\* leafnode(s) and parameter mapping \*/
  6.      $Current\_Rule \leftarrow Current\_Rule * Initial\_Rule$  /\*\* rule formation \*/
  7. Output  $Current\_Rule$
- 

**Fig.7. 18:** Classification rules generation algorithm

In the algorithm, there is a number of leafnodes  $N$  given or considered under a *desired\_Concept*. Firstly, the first leafnode is mapped to the two given boolean parameters  $P$  and  $F$  (i.e. *passed* and *failed*): an operation that generates the first two rules. Subsequently, to obtain further rule combinations, the outcome of the previous mapping is mapped to the outcome of a current mapping to produce the new classification rules. This process is graphically shown in Figure 7.19.

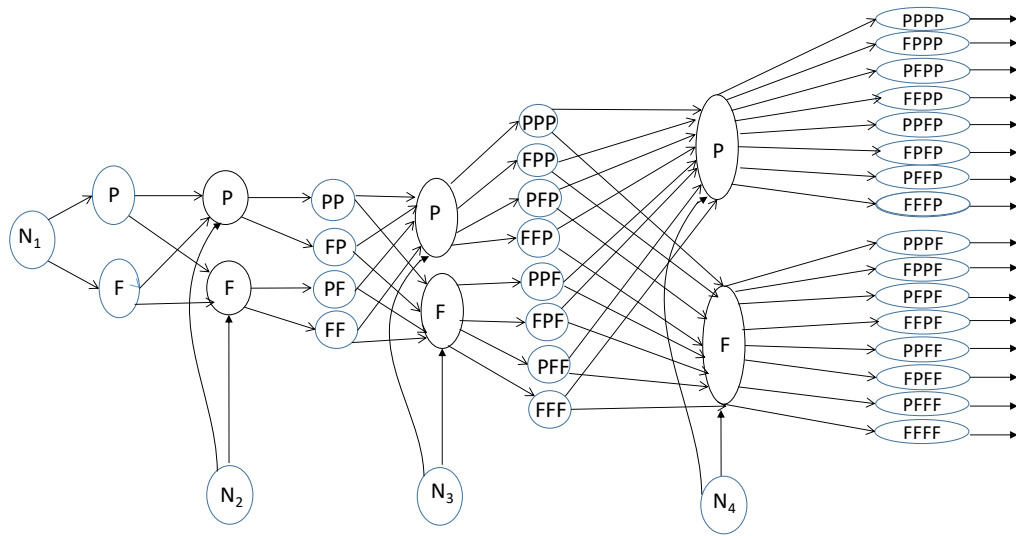


Fig.7. 19: Classification rules formation process

The classification rule formation process is found to be suitable for generating the rules for the two strategies of pre-assessment, which are:

- *Pre-assessment By Immediate Prerequisite Class*, and
- *Pre-assessment By Multiple Prerequisite Classes*

as outlined in this research.

Given the Figure 7.10, now to estimate the total number of classification rules  $R$  for the agent *agModelling* (the classifier) based on the strategy of *Pre-assessment by Multiple Prerequisite Classes*, let us apply the equation as earlier stated in Chapter 4:

$$R = 1 + \sum_{i=1}^k C_i T^{N_{i,j}}$$

Since, the strategy is for a non-regular ontology, the variable  $C_i$  (of the prerequisite parent classes to the *desired\_Concept*) takes a unit value i.e. 1. Thus

for the *desired\_Concept* Union,  $C = 1$  and  $N = 3$

for the *desired\_Concept* Join,  $C = 1$  and  $N = 4$

for the *desired\_Concept* Update,  $C = 1$  and  $N = 4$

for the *desired\_Concept* Delete,  $C = 1$  and  $N = 2$

for the *desired\_Concept* Insert,  $C = 1$  and  $N = 4$

therefore, on a vertical traversal

$$R = 1 + [C_{2,2}T^4 + C_{3,3}T^2 + C_{2,3}T^4 + C_{2,4}T^4 + C_{2,5}T^3]$$

$$R = 1 + (1 * 2 ** 4) + (1 * 2 ** 2) + (1 * 2 ** 4) + (1 * 2 ** 4) + (1 * 2 ** 3)$$

$$R = 1 + 16 + 4 + 16 + 16 + 8$$

$$R = 1 + 60$$

$$R = 61 \text{ number of classification rules}$$

This estimate  $R = 61$  is the number of [*passed* or *failed*] predicate statement that have been combined for the non-regular ontology. multiple class pre-assessments with respect to the number of leafnodes  $N$  considered for the system. Given the equation, the of classification rules  $R$  is determined by the number of leafnodes  $N$  underneath some desired concepts.

## 7.8 Agent agModel: The Store Agent

Updated beliefs are data that are perceived and stored by agents. As mentioned earlier, beliefs can be short-term or long-term for storage of percepts or activities in the system. While other agents in the MAS has short-term beliefs by reason of the fact that perceived percepts are lost when the MAS is stopped, the agent *agModel* (*student*) is the long-term belief base agent configured at the point of the agents' creation, *see Chapter 5, Figure 5.14*. This is for the MAS to store all students' activities which comprised the SQL skills data presented in some part of Chapter 6 and Chapter 7.

## 7.9 Agent agMaterial

This agent performs the last function of the MAS, which is the release materials for students at the end of pre-assessment sessions. As already mentioned, material URLs are released after classification by the classifier agent *agModelling*. Employing the *Pre* and *Post* conditions (Labrou & Finin, 1998), the following are the *Pre* and *Post* conditions of this agent:

*Pre: hasPrerequisite(x, y)[source(self)] //B*

*Pre: hasKB(y, z)[source(self)] //B*

**Pre:** *hasContent*(z, url)[source(self)] //B

**Post:** Adopt a plan with *hasPrerequisite*(x, y)[source(sender)],

Or adopt a plan with *hasKB*(y, z)[source(sender)]

**Post:** ?*hasContent*(z, url)

**Post:** release material url

The **Pre** conditions are the ontological binary relations that are initialised as beliefs *B*. They are the premise in which the classified students' message content from the classifier agent *agModelling* is matched for a plan(s) to be triggered before the release of materials. In the **Post** conditions are *test goals* in the form ?*hasContent*(a, url) in the plan *body*, (Fig. 7.20). Prior to the release of the materials, the *test goals* are used by the agent to query its belief base whether a relation exist that contains the URL links for students after a plan is triggered. From the semantics of speech acts (Labrou & Finin, 1998), the **completion condition** is the effect the learning materials will have on students. As asserted in Manouselis *et al.* (2011), *Chapter 2*, recommended learning is an effort and time taking activity; for students to acquire the requisite skills, the Pre-assessment System was programmed to identify relevant skill needs of students with support on how to achieve them.

```
//learning material

@inner_joinURL
+!has_KB(X, inner_join)[source(agModelling)] : true
  <-.println;
    .println(" You will learn INNER JOIN query statements.
Please use the link for materials:");
    ?hasContentText(innerJoin, IJ_textURL)[o(sql)]; //Test goal
    .println("INNER JOIN query Text Material: ");
    .println(IJ_textURL); .println.
```

**Fig.7. 20:** Agent *agMaterial* use of test goal ?*hasContent* before the retrieval URL materials for students.



## 7.10 The Pre-assessment Sessions

The following section presents and discusses the results gathered from the evaluation of the Pre-assessment System. It comprises the analysis of students' SQL input queries, and students' post-evaluation feedback.

## 7.11 Analysis of SQL Query Statements at Pre-assessment Sessions

From the inspection of the agent TextPersistent beliefs, the gaps that existed in students' construct of SQL query were identified. In a step-by-step analysis, this Section presents students' interaction with the system starting from the submission of their *desired\_Concept*, to the questions they responded to and their SQL query statements, and down to the recommendations made. The analysis looked critically at two selected *Case Studies*, and tried to unravel the possible factors that may be responsible for the learning gaps. Also discussed is the inherent implications of these *Cases* for the teaching of SQL.

### 7.11.1 Case Study I: The UPDATE Desired\_Concept

The student learning target was the Update topic as shown in (TABLE 7.5, S/N. 6).

Thus,

1. **Student's desired\_Concept:** UPDATE.
2. **Inter-agent Communication:** `desired_Concept("update, date(2015-4-7), time(11-3-17)") [source(agSupport)].`
3. **Prerequisite 1:** Delete all penalties who live in the same town as player 44, but keep the data for player 44
4. **Inter-agent Communication:** `quizDeleteSelect("Delete all penalties who live in the same town as player 44, but keep the data for player 44., date(2015-4-7), time(11-3-17)") [source(agSupport)].`
5. **Student's query response:** `DELETE FROM (SELECT * FROM TENNIS_PENALTIES WHERE PLAYERNO = 44`

6. **Inter-agent Communication:** `responseToDeleteSelect("DELETE FROM (SELECT * FROM TENNIS_PENALTIES WHERE PLAYERNO = 44), date(2015-4-7), time(11-9-27)") [source(agSupport)].`
7. **MAS Feedback:** you have NOT Passed the DELETE\_SELECT
8. **Inter-agent Communication:** `failed("The student has NOT passed the DELETE with SELECT question.") [source(agSupport)].`
9. **Prerequisite 2:** Delete all penalties incurred by player 44 in 1980
10. **Inter-agent Communication:** `quizDeleteWhere("Delete all penalties incurred by player 44 in 1980., date(2015-4-7), time(11-9-27)") [source(agSupport)].`
11. **Student's query response:** `DELETE FROM SELECT * FROM TENNIS_PENALTIES WHERE PLAYERNO = 44`
12. **Inter-agent Communication:** `responseToDeleteWhere("DELETE FROM SELECT * FROM TENNIS_PENALTIES WHERE PLAYERNO = 44, date(2015-4-7), time(11-9-58)") [source(agSupport)].`
13. **MAS Feedback:** you have *NOT* passed the DELETE with WHERE.
14. **Inter-agent Communication:** `failed("The student has NOT passed the DELETE with WHERE question.") [source(agSupport)].`
15. **MAS Recommendation:** *URL* recommendation to learn both prerequisite concepts in ***DELETE***.

### 7.11.2 Case Study II: The JOIN Desired\_Concept

In this Case Study, the student's intended learning concept was the Join ( *TABLE 7.5, S/N. 10*). Thus,

1. **Student's desired\_Concept:** JOIN.
2. **Inter-agent Communication:** `desired_Concept("JOIN, date(2015-9-16), time(11-01-15)") [source(agSupport)].`
3. **Prerequisite 1:** Set the number of sets won to zero for all players resident in Stratford.
4. **Inter-agent Communication:** `quizUpdateSelect("Set the number of sets won to zero for all players resident in Stratford., date(2015-9-16), time(11-01-15)") [source(agSupport)].`

5. **Student's query response:** `SELECT * FROM TENNIS_MATCHES`
6. **Inter-agent Communication:** `responseToUpdateSelect("SELECT * FROM TENNIS_MATCHES, date(2015-9-16), time(11-3-16)") [source(agSupport)].`
7. **MAS Feedback:** you have NOT Passed the UPDATE\_SELECT.
8. **Inter-agent Communication:** `failed("The student has NOT passed the UPDATE with SELECT question.") [source(agSupport)].`
9. **Prerequisite 2:** Change the value F in the SEX column of the PLAYERS table to W (women).
10. **Inter-agent Communication:** `quizUpdateWhere("Change the value F in the SEX column of the PLAYERS table to W (women)., date(2015-9-16), time(11-3-16)") [source(agSupport)].`
11. **Student's query response:** `UPDATE SEX FROM P WHERE SEX = 'F' TO SEX = 'W'`
12. **Inter-agent Communication:** `responseToUpdateWhere("UPDATE SEX FROM P WHERE SEX = 'F' TO SEX = 'W'`
13. **MAS Feedback:** you have NOT passed the UPDATE with WHERE.
14. **Inter-agent Communication:** `failed("The student has NOT passed the UPDATE with WHERE question.") [source(agSupport)].`
15. **MAS Recommendation:** URL recommendation to learn both prerequisite concepts in **UPDATE**.

**TABLE 7. 5: SUMMARY OF CORRECT AND INCORRECT ANSWER RESPONSES**

**NB:** Passed  $\equiv$  1 and Failed  $\equiv$  0

S/N	Desired Concept	Prerequisite leafnode N & Time of Quiz Display (HH-MM-SS)	Time Student Responded (HH-MM-SS)	Time Spent on Task (HH-MM-SS)	Classification of Students' Skills [0 or 1]
1.	INSERT	SELECT_WHERE 12-10-23	12-13-54	00-03-31	0
		SELECT_ALL 12-13-54	12-13-59	00-00-05	0

2.	INSERT	SELECT_WHERE 12-14-40	12-14-46	00-00-06	1
		SELECT_ALL 12-14-46	12-15-30	00-00-44	1
3.	DELETE	INSERT_SELECT 12-17-38	12-22-18	00-04-44	0
		INSERT_VALUE 12-22-18	12-22-37	00-00-19	0
4.	INSERT	SELECT_WHERE 12-29-43	12-32-04	00-02-21	0
		SELECT_ALL 12-32-04	12-33-06	00-01-02	0
5.	UNION	FULL_OUTER_JOIN 12-42-14	12-59-10	00-16-56	0
		INNER_JOIN 12-59-10	13-01-19	00-01-29	1
6.	UPDATE	DELETE_SELECT 11-08-54	11-09-27	00-00-33	0
		DELETE_WHERE 11-09-27	11-12-10	00-02-33	0
7.	UPDATE	DELETE_SELECT 11-11-31	11-12-10	00-00-39	0
		DELETE_WHERE 11-12-10	11-14-14	00-02-24	0
8.	UNION	FULL_OUTER_JOIN 11-28-48	11-28-56	00-00-08	0
		INNER_JOIN 11-28-56	11-29-35	00-00-39	0
9.	UNION	FULL_OUTER_JOIN 11-29-48	11-31-43	00-01-55	0
		INNER_JOIN 11-31-43	11-34-04	00-02-21	0
10.	JOIN	UPDATE_SELECT 11-01-15	11-03-16	00-03-01	0
		UPDATE_WHERE 11-03-16	11-05-01	00-01-45	0
11.	INSERT	SELECT_WHERE 11-11-47	11-12-57	00-01-10	1
		SELECT_ALL 11-12-57	11-13-51	00-00-54	1

## 7.12 Findings from The Pre-assessment Exercise

From the *Case Studies*, it is apparent that there are learning gaps in the students' SQL query knowledge which might not have been known to the students themselves. This is evident from the fact that they thought they were prepared for the *desired\_Concept* they entered to learn. They believed that they could answer the prerequisite questions to the(ir) *desired\_Concepts*. These were assessed to have *NOT Passed* the prerequisites in both *Case Studies I* and *II* (see lines 7 & 13), respectively. These are irrespective of the time spent on tasks or by the number of attempts (e.g. twice) made. In all of the pre-assessment *cases*, the System recommended the learning of the appropriate materials according to the performance of each of the student.

## 7.13 Implications for Teaching

Programming is not an easy subject to study (Lahtinen, Ala-Mutka & Järvinen, 2005; Ala-Mutka, 2004). Particularly for this study, SQL programming can be very difficult because of the activity involved in translating a natural language question into a semantically correct SQL expression (Sadiq *et al*, 2004). Such underlying factors have influenced a number of systems research on ways to improving students' SQL coding skills (e.g. Wang & Mitrovic, 2002; Kenny & Pahl, 2005; Sadiq *et al*, 2004). As given in Prior (2003) mapping from a problem statement describing what information is required from the database into an appropriate SQL statement is not easy.

From the analysis of results and findings in students' SQL query constructs from the cases being reviewed in the preceding sections, students may have inherent gaps in SQL query constructs from previously learned SQL concepts without realising it. Tutors need to understand this: To handle courses with uttermost diligence so as to take students through learning with emphasis on the difficult or technical constructs (such as the use of operators, SQL query keywords, and subqueries) where misconception may arise.

Considering the *Case Study I* (Section 7.11.1), the pre-assessment problem that was posed to the student was a sub-query problem — a *DELETE SELECT* (line 3 or 4). The student was able to decipher that the problem was a sub-query task but encountered difficulty in the process of organising the query statement. From the student's SQL query statement, the *main* part of the query missed out on:

- the *table-name*,
- the *where* clause,
- the *column\_name*, and
- the *operator*.

On the *sub-query* part, the `Select All` (“*SELECT \* ...*”) query expression was the student’s response (*line 5*) in the *case studies*, sections 7.111 and 7.11.2 respectively. Though on the question (*line 3 or 4*), there was the term “*all penalties*”. This does not imply all fields in the table. So this may have put the student in a tight situation to infer that this meant *all the columns or fields* in the table. But this only refers to the *penalties* field. Further, on prerequisite 2 (*line 9 or 10*) where the problem was a `Delete Where` task, the student was aware that this is not a sub-query task. However, the query (*line 11 or 12*) also missed out on the following information:

- *table-name*, and
- *specified column\_name*;

instead the (“*SELECT \* ...*”) was also used to “*select all*” the column-names.

In *Case Study II* (Section 7.11.2), the first pre-assessment task was also a *sub-query* problem (*line 3 or 4*). Unlike in *Case Study I* where the student was able to decipher that the problem was a subquery problem (even when the system supported some pre-assessment problems with hints on the type of problem), in this *Case Study*, the student was unable decipher this. The SQL query submitted by the student was as `Select All` (“*SELECT \* ...*”) statement (*line 5 or 6*). Further to the next prerequisite assessment (*line 9 or 10*), the student had difficulty by submitting the `UPDATE` query statement that had a *field* or *column\_name* before the supposed *table\_name* (which the student stated as “*P*”) and also using the word “*TO*” in the query (*line 11 or 12*). Shown below is the student’s answer:

```
UPDATE SEX FROM P WHERE SEX = 'F' TO SEX = 'W'
```

against the correct and expected answer in the System

```
UPDATE TENNIS_PLAYERS SET SEX = 'W' WHERE SEX = 'F'
```

As shown above in the student's query, the statement missed out on the *SET* keyword for the UPDATE query.

This analysis has revealed in detail the area of difficulties faced by students. It also underscores the area in which tutors of SQL can give greater attention. From the *Case Studies*, it could be stressed that some students are yet to have a good grasp of SQL query syntax. SQL syntax has a defined format and structure that can be adhered to when constructing queries. This format gives the order of precedence of SQL *keywords*, *table\_names*, *column\_names* and their *operators* in a query statement.

The Pre-assessment MAS has not only identified *gaps* in learning but has also identified skills *gained* by student as described by the modelled parameters and the logic of classification in Chapter 4. Knowledge gain was identified in some of the pre-assessment cases based on the regular ontology of 2 leafnodes across all parent class nodes (see TABLE 7.5). In one of the data stored, the student's *desired\_Concept* was the INSERT topic. After the pre-assessments on the `Select Where` and the `Select All` query, the student was adjudged "*Passed*" and recommended to study the INSERT desired topic entered.

The TABLE 7.5 is a collection of all the data of the activities that took place in the System. This include the desired concepts, the time spent on each task, and the class of the answers submitted as assessed by the Pre-assessment System. From the data in TABLE 7.5 two cases of recommendation for the *desired\_Concept* occurred in the survey (described as *positive ability* in Chapter 5); one case of a *passed* pre-assessment (described as *partial ability*); and all others cases of *failed* pre-assessment, described as *negative ability*.

As defined in the FOL syntax (Chapter 4) during the specification of the classification process, every *failed* concept is recommended for learning via a URL link to the relevant material; and for all *passed* concepts, the student learns his desired concept (which are the leafnodes to the class node) from relevant URL links too. The *failed* concepts are equivalent to the class of *0s* and the *passed* concepts the class of *1s* as

analysed in TABLE 7.5. From the data, the percentage summary of the *Passed* leafnodes concepts against the *Failed* leafnode concepts is shown in Figure 7.21.

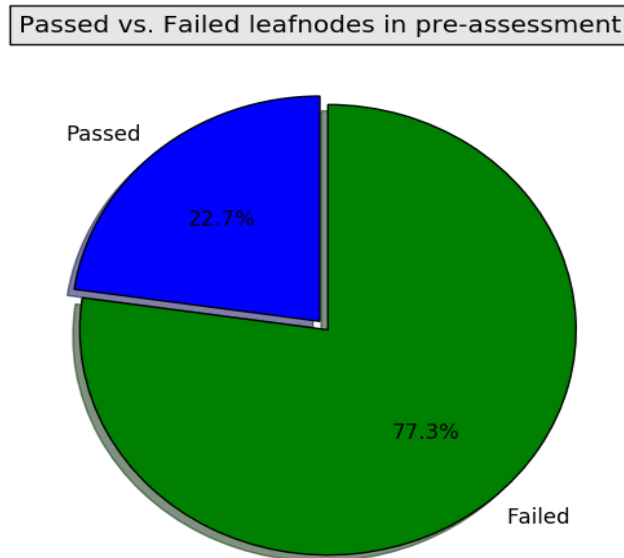


Fig.7. 21: Percentage of number of *passed* vs. *failed* leafnode concepts

As stated in Chapter 5, abilities of students can be further classified into: 1) *positive ability* when all SQL answer queries are all *passed*; 2) *partial ability* when there is a mix of both *Passed* and *failed* SQL query constructs; and 3) *negative ability* when all SQL queries are assessed as *failed*. The Figure 7. 22 represents the details of these abilities.

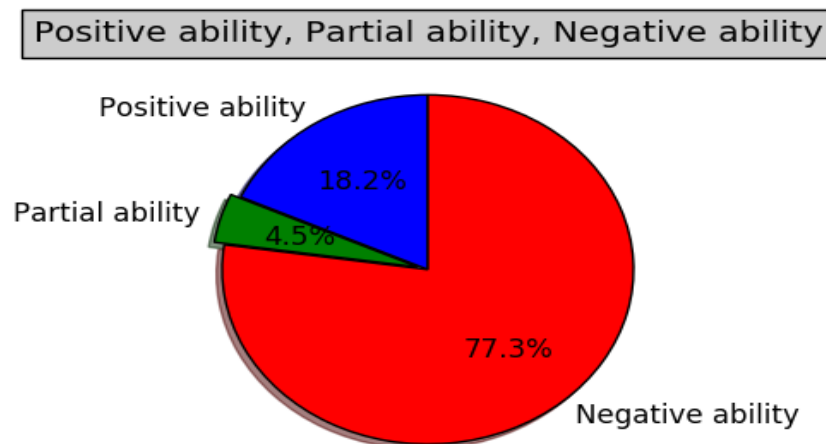


Fig.7. 22: Percentage of students' abilities.



Recall that the Pre-assessment MAS also keep records of *time spent* on tasks by students in its *TextPersistent BB* agent. These beliefs were examined to understand whether *time* was a factor and had any influence on students' performances, on each pre-assessment task. In the TABLE 7.6 is the boolean values [1 or 0] to visualise the *classification* of pre-assessment outcomes against the *time spent* on tasks by students using linear regression. From the data, students' performances have not been influenced by time: the longer time-length spent on tasks did not increased students' chances of remembering or overcoming their difficulties in SQL code constructs. The visualisation of the binary classification is given in Fig. 7.23 after the data was split: 50% training and 50% test, respectively.

TABLE 7. 6: TIME-INDEPENDENT VARIANT STUDENTS' PERFORMANCE ANALYSIS

Time spent (mm.ss)	Boolean classification	Time spent (mm.ss)	Boolean classification
3.31	0	2.33	0
0.05	0	0.39	0
0.06	1	2.24	0
0.44	1	0.08	0
4.44	0	0.39	0
0.19	0	1.55	0
2.21	0	2.21	0
1.02	0	3.01	0
16.56	0	1.45	0
1.29	1	1.10	1
0.33	0	0.54	1

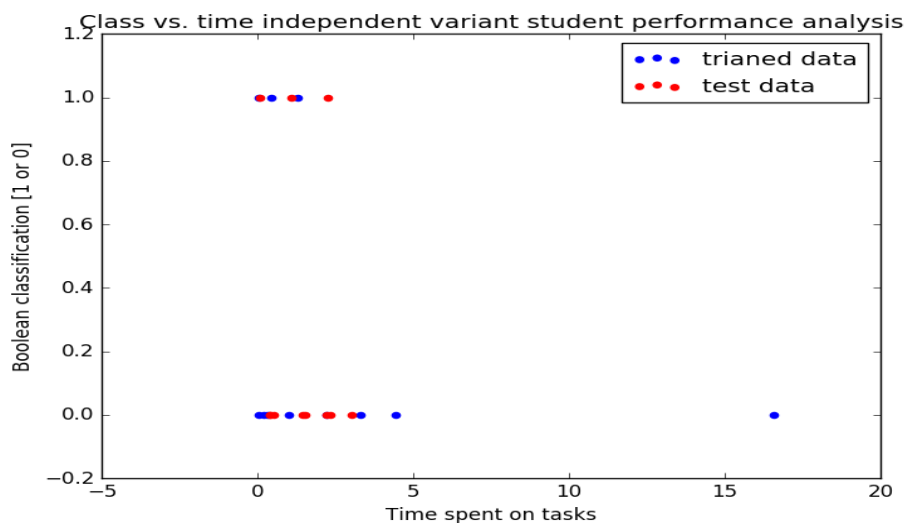


Fig.7. 23: Time-Independent Variant Student Performance Regression Analysis based on the data in TABLE 7.6.

Based on the Figure 7.23 the average time spent on the tasks that were *passed* and those *failed* are largely between 0 and 5 minutes, with one outlier on the 0 class. One of the objectives of this regression analysis was to make predictions, but based on the small amount of data collected, reliable prediction cannot be projected.

Recall that in Chapter 4 it was stated that the *passed* and *failed* predicate parameters were devised not only for the agent classification of students but to also provide increased reinforcements to students during their pre-assessment feedbacks. From the experimental survey with students and the observations made during the pre-assessment sessions, negative reward i.e. a *failed* feedback does increase reinforcement. When some students noticed they had negative feedbacks due to incorrect SQL queries, they immediately wanted to have another attempt, to get their SQL queries right. Like positive rewards for correct answers, negative rewards for incorrect incorrect can instigate reinforcement and did provide positive reinforcements.

### **7.14 Relevance of Chunking in the Pre-assessment System**

Students learn best by *Chunking* of unit of lessons (Casteel, 1988; Anderson, 2008). From the evidence in the students' skill data and the time lapse spent by some students on task, this thesis concurs to the assertion of Prior (2003) that SQL is difficult, and not easy to learn. As stated in Sadiq *et al.* (2004), and as clearly observed, this was because of having to translate a natural language problem into the logic of SQL queries. Thus, the optimal strategy to organise formative assessment materials for students in SQL is by applying the principle of *Chunking* that will enable students to focus more time and attention to the smaller units of the recommended learning materials after their pre-assessments. Because organising a very large number of units of lessons for pre-assessments can potentially lead to task overload from large amount of learning materials being recommended in the event that several pre-assessments are failed. From the survey, students stayed on tasks and studied their recommended materials, as well as having repeated attempts on already *failed* attempts.

## 7.15 System's Post-Evaluation Survey

The aim of the Pre-assessment System of this study as stated earlier was to identify gaps in students' learning and to devise a strategy through agent classification learning on how to assist students in filling the gaps. From the data presented in *Chapter 6, Section 6.3* and the analysis of the preceding *Sections 7.11 – 7.14*, the study has revealed that 77.3% of students in the survey have inherent skills gap in their construction of SQL queries. In the following Section, the Pre-assessment System post-evaluation survey data is presented and discussed. The data covered students' perception of the Pre-assessment System, the pre-assessment sessions, and about students previous SQL studies. A 17 item structured questionnaire was used to collect data, including demographic data.

### 7.15.1 Student Course Distribution Data

With questions 1 and 2 (*Q1 & Q2, see Chapter 6, and Appendix B.B1*) course distribution and the level of study of the student participants that took part in the survey was collected. As shown in the TABLES 6.2 and 6.3 of Chapter 6, 29% represented students in *Software Engineering*; 43% in *BSc Information Tecnology with Business Studies*; and 14% in *MSc Database Professional and Enterprise System Professional*, respectively. The survey comprised of students from both undergraduate and postgraduate studies with 71.4% being *Second Year* students; and 14.3% *First Year* and *MSc* students, respectively (TABLE 6.4).

### 7.15.2 User Perception of The Pre-assessment System and Sessions

Questions 3 – 9 (*Q3 -Q9*) investigated students' view about the System's fitness-for-purpose and responses were gathered as qualitative data (TABLE 6.5, Chapter 6). Question Q3 sought students' opinion on whether the system was useful. Responses showed that 14.3% **Strongly Agreed**, 71.4% **Agreed**, while 14.3% were **Undecided**. In Question Q4, it was asked whether the System helped to recall previous SQL learning experiences. The responses received are 57.1% **Agreed** and 42.9% **Strongly Agreed**. Q5 sought to find out whether the system supported their learning of SQL, 28.6% of the participants **Strongly Agreed**, 57.1% **Agreed** while 14.3% were **Undecided**. The

survey also wanted to know whether the participants were not familiar with SQL. The response gathered revealed that participants have studied SQL previously: with 14.3% *Strongly Agreed*, but 57.1% *Disagreed* and 28.6% *strongly disagreed* respectively that they are “NOT familiar with SQL”. By implication, 85.7% *Agreed* and believed they were well acquainted with the concept of SQL and database queries. In terms of MAS system directing the course of the pre-learning assessment, 85.7% of the participants *Agreed* that they were guided by the system, while 14.3% were *Undecided*.

From Questions *Q9 – Q11*, with 42.9% *Strongly Agreed* and 57.1% *Agreed*, it was made known that participants understood the design purpose of the system, and acknowledged the role of the researcher in facilitating the pre-assessment sessions. The latter is for the researcher’s reflection on the part he took at the sessions.

In *Q12*, while the data revealed that 14.3% *Strongly Agreed*, 57.1% *Agreed* that the session was a good learning experience; 14.3% *Disagreed*. In *Q13*, 28.6% *Strongly Agreed* and 57.1% *Agreed* that the sessions were well organised; 14.3% were *Undecided*.

### 7.15.3 Open-Ended User Feedback

Using *open-ended* entries from questions 14 to 17 (*Q14 – Q17*), diverse views about the pre-assessment sessions or the System that could not possibly be captured by the *closed-item* questions in *Q3 -Q13* were elicited. From these responses, some student users found the pre-assessment sessions and system satisfactory while others made comments on important issues that are salient enough to improve usability design and usage experience in further work.

In **TABLE 6.7** students’ view were sought on: *what was least interesting about the sessions?* One view was that

**“Lack of equipment available. Session was slow.”**

The **AOP** language for developing the Pre-assessment System is Jason AgentSpeak, a logic based programming language. So, prior to the various pre-assessment sessions, volunteer participants were scheduled for different times to evaluate the System. But

in the course of a participant's use of the system, some participants encroached into the time schedule of another participant. This was due to the time some participants needed to understand their questions, understand the data on the MySQL database server, and construct their SQL queries.

Initially, the agent based Pre-assessment System was developed to connect to the MySQL Workbench database server. Review of System development after the prototype had the Pre-assessment System disabled from the database server. This is because of the need for one system to host the database, and another for the Pre-assessment System. Thus, in the course of the participants' usage of the System, two systems were made available: one opened for the data on the TENNIS\_DATABASE and the other for taking the pre-assessment exercises. In that regard, the issue of

**“We only had one monitor to do the work on.”**

was addressed.

Also on the view in TABLE 6.7 that

**“The system is not quite flexible and does not allow trial and error terms. One small error led into decision that we need to learn the module. ...”**

Like most formative assessment or self-diagnostic systems that assesses knowledge, the Pre-assessment System is programmed to take in an input or percept when submitted, then assessment, and then next question. As result, some participants in the study who felt the need to retake their assessment, did so as many times as they needed. The Pre-assessment System is flexible and will allow the pre-assessment about a given *desired\_Concept* to take place over and over again. This is recorded in the skills data collected and showed some students took their assessment twice on the same module. The views from the TABLE 6.9 that participants

**“Having to switch between three different windows to operate the system”**

has to do with the built-in MAS output console and the input window for participants SQL queries answers. Recall that, agents are components that can be situated in some [student] environment in order to fetch or observe percepts. As a result, the input window was configured for *open-ended SQL queries* using the CArtaGO artifact. Participants' text-inputs are perceived by the MAS through this artifact, and after processing by the MAS, outputs are displayed through the Jason built-in MAS output console. Future work will consider one window for both input and output. One other important view from TABLE 6.9, is that

**“The system covered limited SQL statements so when more are added I think it will be more interesting.”**

This is what the strategy of *Pre-assessment By Multiple Prerequisite Classes* has addressed. Where more unit of lessons are added to parent class nodes or modules (Fig. 5.4, Fig. 7.10), and also, pre-assessment across multiple class nodes as specified in the ontology tree.

In TABLES 6.6 and 6.8, participants expressed satisfaction on the concepts of pre-learning and teaching through the Pre-assessment System where they have to learn what is appropriate. This is one view from TABLE 6.6, *entry no. 3*, which states

**“It is actually a good objective, we will learn what exactly we need to learn. Because sometimes tutor[s] teach something which is redundant since some people already understand it well.”**

This aligns with one of the objectives of this System: To avoid putting every students in the same starting block on the learning ladder. At any given level the student can build up the ladder. While this System would allow students that has solid understanding of some concepts already to progress to the next or higher level of learning. Those with misconception and difficulty would be assisted by the System to identify the weaknesses in their learning, and be assisted to fill those gaps in the absence of the tutor. When what is already known by say *Student X* is being taught all over again with *Student X* present, this becomes “*redundant*” to that student.

The purpose of modelling students' skills for adaptive learning in this work is for the intelligent system and the course tutor to give optimum support for improved performances. As required of a typical system of diagnosis and fault detection (in students' cognition), the Pre-assessment System through classification reasoning has identified and recommended learning appropriate for participants in this evaluation exercise.

## 7.16 Summary of Chapter

The Pre-assessment System, its broad goal, which is to identify gaps in learning and classification process of learning has been presented in this Chapter. The Chapter described the Pre-assessment System as a reactive system of five interacting agents. Where the agent *agSupport* is the pre-assessment agent that uses *!achievement goals* – the state an agent wants to accomplish – for the pre-assessment of knowledge. Each *!achievement goal* corresponds to each leafnode in a given ontology tree. For the recommendation of appropriate learning materials, classification is first carried out based on the *passed* or *failed* boolean parameters predicate decision statements from agent *agSupport*. The agent *agModelling* classifies students before the release of learning material by agent *agMaterial*. This Chapter also discussed algorithms, and generation of classification rules. The generation of the classification is based on the FOL rules: the formal reasoning representation (*from Chapter 4*) and its application for the realisation of the classification plans in the agent *agModelling*.

Two strategies, namely: *Pre-assessment By Immediate Prerequisite Class* and *Pre-assessment By Multiple Prerequisite Classes* that evolved from the *Pre-assessment Mechanism* were also presented. While the data collected from the implementation of the former was analysed and discussed; the chapter had the implementation of the latter discussed. Based on the results from the experimentation and background literature on the learning of SQL, the position of this thesis is that the educational theory of *Chunking* (Casteel, 1988; Anderson, 2008) which is to present tasks of learning to students in smaller units, can support students to succeed in their learning of SQL. This is based on the data gathered in Chapter 6 in which 77.3% of the unit of lessons (leafnodes) were not passed, (*see TABLE 6.2, and Fig. 7.21*). Yet students stayed on

tasks to study recommended materials. From the foregoing, organising and allocating units of lessons in smaller quantities has enabled students to remain on tasks to study recommended materials. When one desired learning concept is successfully completed, another desired concept can be attempted for learning. In the next Chapter 8, the conclusions for this study shall be presented along with its contribution to knowledge, and future work.



# Chapter 8

## Conclusions and Future Work

This study has demonstrated pre-assessment and learning path recommendation strategies like a face-to-face tutor would do so as to boost competency level of students before the start of a new lesson. The thesis covered two strategies of pre-learning assessment using an agent based approach in order to fill the gaps in learning and support further learning. In this work, the multiagent *Pre-assessment System* was investigated, developed and evaluated: as a System aimed at identifying gaps in students' learning and making learning materials recommendation to fill-in the gaps. From this implementation and evaluation of data, it has been shown that the *Pre-assessment System* can perform its classification function in accordance to its rule based knowledge representation process in which students' prior learning is pre-assessed and materials are recommended for learning. This has followed a *Pre-assessment Mechanism* that depicts the process or strategy of pre-assessment of lower concepts in order to measure what has been learned successfully by a student before the start of a higher or *desired\_Concept* intended for learning. The Pre-assessment System's investigation began by identifying the research problem as a classification problem in a learning domain in which students' skills set would be collected and categorised for learning material recommendation.

### 8.1 Research Development Approach

The research approach to this study is dual in nature, namely: rule based classification procedure, and agent oriented software engineering through the *Prometheus* methodology (Padgham & Winikoff, 2004) for the Pre-assessment System design. *Prometheus* is a methodology for developing intelligent agent systems and has a customised tool known as the Prometheus Design Tool (PDT) for designing *BDI* agents. The PDT has been used in the design specification and analysis of the pre-assessment multiagent system as well as its rule based representation, *as outlined in*

Chapter 4. The agents were developed with individual responsibility and to function as components that make up a whole sum. As with an organisation, its organisational parts must be able to interact cooperatively, with individualised roles in order to realise its design objective.

To solve and answer the research question, a structured hierarchy of learning was outlined in the domain of SQL. The domain was then analysed after its definition as a TBox with a description logic (DL) language. The analysis presented the inter-relations between the ABox instances i.e. concepts, individuals and roles in accordance to the given learning structure (Fig. 5.1) which enabled students to have their prior knowledge assessed. Thereafter, they can progress from one lower level of learning to the next higher level, *see Chapter 5*. After implementation, the System evaluation showed that the system diagnosed students' state of SQL knowledge, captured their areas of difficulty and pointed them to learning material to close the gaps in their learning. Another benefit of the of the Pre-assessment System is that the learning activities are stored, especially the SQL queries, and these can be teaching resource for the tutor. The tutor can use this resource to unravel the the technical difficulties or challenges faced by students, and also, pay greater attention to these challenges during teaching.

The following is a recap of the objectives of this research as stated in Chapter 1 and how they have been addressed:

- *To investigate a systematic way of identifying gaps in students' knowledge which may hinder them in their next stage of learning. This is to allow students to self-diagnose any gaps on their previous learning before the start of a new module.* In that regard, the research team deciphered that gaps could be identified between two ends: which are a start-point and an end-point of pre-assessment. This led to the flow-chart of the Pre-assessment Mechanism (*Chapter 4, Fig. 4.19*) in which a student could enter a *desired\_Concept* (i.e. the start-point), go through some prerequisite assessments to the end of the *leafnodes N*, get result(s), and have learning recommended.

- *To build a domain ontology of related concepts and use declarative logic based representation in the system in the process of learning gap identification prior to the start of a higher and desired learning by students.* A domain subject of learning was needed as the content of the system. The SQL learning domain was chosen. The choice of SQL was based on the good enrolment records of students in DB. Which was also envisaged would produce a good number of volunteer participants for the survey. Then a hierarchy of topics (concepts) as a learning structure was developed based on the teaching notes of DB lecturers in the department of computing. This led to the definition of the ontology: concepts, individuals and their relations using a DL language (*Chapter 5*).
- *To investigate the communication of ontological concepts in the system in the process of identifying gaps in students' learning.* As a multiagent based system, agent must communicate. The thesis looked into the communication of knowledge: from environmental percepts, to decision statements, and to the ABox assertive knowledge in their unary and binary predicates. Then chose the *tell*, *askOne* and *achieve* performatives for inter-agent communication in system using the *.send()* standard internal action (*see MAS implementation in Chapter 5, and discussion in Chapter 7*). This is against the *.broadcast()* standard internal action whose message in some occasions didn't trigger agent to fire their plans .
- *To develop the tools that allow the system to recommend supplementary study materials to close the gaps in their current learning.* This covers the design (*Chapter 4*) and implementation (*Chapter 5 & 7*) of the Pre-assessment System.
- *To evaluate the effectiveness of the system by assessing how effective it is in helping real students improve their learning.* This is where the Pre-assessment System was assessed for fitness of purpose by students. Students used the system, and self-diagnosed their learning. Where students made errors and *failed* a concept, material URLs were recommended. But where all pre-assessments are *passed*, students were recommended for their *desired\_Concept*, (*See data in Chapter 6*). They opened the links and studied materials.

## 8.2 Contributions to Knowledge

In summary, the following are the contributions of this research:

1. *Identifying gaps in students' learning using a devised Pre-assessment Mechanism:* As stated in the objectives, *Chapter 1*, the study has investigated systematic strategies to identifying gaps in students' learning. The realisation of this objective comprised two identified strategies: *Pre-assessment By Immediate Prerequisite Class*, and *Pre-assessment By Multiple Prerequisite Classes* that originated from the *Pre-assessment Mechanism* in *Chapter 4*. The educational principle of *Chunking* (smaller unit of lessons) was applied as the underlying principle and optimal strategy in developing the agent based e-learning system. The System has supported students to identifying gaps or gains in their current learning and also making recommendation to close the gaps. This is in a subject domain that is ascertained by researchers in literature as "difficult and challenging".
2. *Goal specification using agent oriented software engineering for developing e-learning system.* This is from requirement specification, to agent goals, to functionality specification, to agent role grouping, interaction, protocols and capabilities in the development of the intelligent agent based e-learning system, *see Chapter 4*.
3. *Use of description logic syntax for defining an ontology of a learning domain.* The study developed an ontology in a learning domain as the content of the agent based multiagent system using a DL language. The DL defined the TBox terminology and named the ABox instances in the domain of SQL. Given the form of a unary predicate  $p(a)$  and binary relation  $R(a, b)$  or  $p(a,b)$ , a collection of agent beliefs (also known as knowledge in first order logic) were modelled as *ground* facts. These facts have been used by agents in the system for communication of knowledge in the diagnosis of students' prior skills and during recommendation for appropriate learning materials, *see Chapter 5*.
4. *Modelling classification features with logic based representation (or architecture) for agent plans for the recommendation of appropriate*

*knowledge-level learning materials*. Based on the boolean state: *passed(N)* and *failed(N)* parameters and the *desired\_Concept(D)*, first order logic notations were used to define the classification rules that categorised students' skills. The classification rules are a collection of axioms that is dependent on the number of leafnodes underneath a given *desired\_Concept(D)*, see Chapter 4, and discussion on implementation in Chapter 5 & 7.

### 8.3 Limitation of The Study

As with most research, this study is not without any challenges. This centres around the small number of volunteer-participants in the survey, and the system constraints with the Jason AgentSpeak language.

#### 8.3.1 Volunteer Population Sample of the Study

This is the aspect of this study where only 7 volunteer participants were recruited for the system evaluation in a survey exercise that spanned across four academic semesters. This number is well below the recruitment projection made at the early stage of this study by the research team.

#### 8.3.2 System Constraint with Jason AgentSpeak Language

Aside from keeping to the educational principle of *Chunking* (Casteel, 1988; Anderson, 2008) in the development of the Pre-assessment System, it was also observed that Jason AgentSpeak language had some limitation in completing the execution of the plan corresponding to the fifth or more *leafnodes*  $N \geq 5$  in the sequence of prerequisite assessment, e.g. Figure 5.4. This is where the agent plan that needed to assess SQL query answer of the fifth pre-assessment leafnode i.e.  $N = 5$  was not triggered. This constraint halted the adoption of the next *!achievement goal* by pre-assessment agent. Yet the agent's *Mind Inspection* revealed that the agent received the required percept for such agent plan to be triggered from the sending agent.

### 8.3.3 Alternative Languages of Implementation

Jason has been used in this work to test our model theory of agent based system for pre-assessments in students' learning after the analysis of a number of agent oriented programming languages (AOP) and platform (see Section 3.9 and Table 3.1). This is because Jason was readily available as open source language that met all our implementation requirements. From implementation, our model theory of logic based rules for classification reasoning in pre-assessment were verified and validated. Nonetheless, the following highlights a few AOP languages and platforms that are suitable alternatives to Jason:

- Jack: Jack is a language with a BDI mental model. With its integrated graphical environment, the Jack Development Environment can be used to develop the pre-assessment multiagent systems or distributed agent application across multiple network devices. As shown in Figure 3.9, the Prometheus agent analysis and design methodology supports the generation of skeletal Jack code for straight-forward implementation on Jack™.
- Jade middleware architecture: Jason runs on Jade based on the “Jade” infrastructure. As a middleware platform, Jade can be used to develop and distribute the pre-assessment system on different network hosts. Jade supports semantic web languages such as XML.
- Jadex language and middleware platform: Jadex can also be applied in the development of distributed intelligent agents on the BDI paradigm. Besides, Jadex framework is realised when agents sit on the Jade middleware infrastructure, use it and run on it. Like Jade, Jadex also supports the XML web semantic technology.

### 8.4 Further Work

The Pre-assessment System has been developed with a group of five agents, but with one agent in charge of the pre-assessments of all the leafnodes. Depending on the number of concepts and leafnodes, future research intends to look into the development of more number of agents (swarm of agents), so as to have one agent per

concept or leafnode in the conduct of pre-assessment. This is likely to resolve the system constraint encountered in Jason.

Two strategies of pre-assessments have been identified in this study. Further work will be to conduct more surveys, collect more data, and then compare both strategies so as to evaluate which is the better strategy supporting students through prerequisite assessment for further successful learning.

The Pre-assessment System has operated a single tasking mode. Further investigation would be to look into multi-tasking approach for parallel percept observation, pre-assessments and classification. One way to achieve this is through a web launch of the Pre-assessment System.

Hard-coding training examples for skills classification can be cumbersome when a large number of nodes are considered for pre-assessments. Basically, this is when the boolean predicate parameters are being mapped to every leafnode concept that are included in a pre-assessment activity. In future work, multi-agent learning would be an area to be investigated in order to have agents compute and produce their own classification plans or rules.

Students' performance score was not considered in this system development. In future work use of performance score is an area to be considered. Thus, using the outcome [0 or 1] of students' performance on every leafnode, performance scores could be rated against certain threshold values. Below a given threshold, agents could direct students to revisit a previously attempted leafnode question.

The data drawn from the System survey has been small. Future work will look to gather more data over a large population sample of databases SQL students, so that further regression analysis can be carried out in order to predict the trend of SQL learning by students from time to time.

Jason is a programming language with syntax structure in a Prolog-like syntax. Jason agent communicates semantic literals (unary or binary) as demonstrated in this

research. These are literals that are in first-order logic representation. To this effect, further work will be to explore the connection of agent based system to ontology repositories from where agents can make sense of the data to query and update the repository.

### 8.4.1 Recommendation

The recommendation for future implementation in order to support students' successful learning of SQL are:

1. SQL formative assessment systems should be developed for practice such that DB tutors can have access to students query constructs in order to inform improved teaching methods when tutors see the difficulties faced by students in their queries.
2. Prior learning diagnosis should become part of intelligent learning systems. That is, there should be pre-learning diagnosis before the commencement of a new or desired learning by students.
3. Students should not be overloaded with practice of prior learning assessments. This means, the educational principle of *Chunking* should be considered and employed in the organisation of prior learning assessments.
4. Learning of SQL syntax structure, relational algebra and natural language processing should be prerequisites to SQL coding. Where necessary students should be well acquainted with the maths of set theory and its operators, and decomposition of natural sentence into FOL form or notation.
5. The strategy of prior learning assessments, classification and recommendation of learning materials to fill-in the gaps in students' learning should be adopted in the development of SQL intelligent tutoring and recommender systems before the learning of a relatively desired or higher concepts.



# References

Abelló, A., Rodríguez, M. E., Urpí, T., Burgués, X., Casany, M. J., Martín, C., & Quer, C. (2008, July). LEARN-SQL: Automatic assessment of SQL based on IMS QTI specification. In *Advanced Learning Technologies, 2008. ICALT'08. Eighth IEEE International Conference on* (pp. 592-593). IEEE.

Abdullah, A. L., Malibari, A., & Alkhozai, M. (2014). Students' performance prediction system using multi-agent data mining. *International Journal of Data Mining & Knowledge Management Process (IJDKP)* Vol.4, No.5. Pp.1-20.

Ala-Mutka, K. (2004). Problems in learning and teaching programming—a literature study for developing visualizations in the Codewitz-Minerva project. *Codewitz Needs Analysis*, 1-13.

Alexakos, C. E., Giotopoulos, K. C., Thermogianni, E. J., Beligiannis, G. N., & Likothanassis, S. D. (2006). Integrating e-learning environments with computational intelligence assessment agents. *World Academy of Science, Engineering and Technology*, 19, 117.

Almendros-Jiménez, Jesús M. "A prolog-based query language for OWL." *Electronic Notes in Theoretical Computer Science* 271 (2011): 3-22.

Anderson, T. (2008). *The theory and practice of online learning*. Athabasca University Press.

[http://biblioteca.ucv.cl/site/colecciones/manuales\\_u/99Z\\_Anderson\\_2008-Theory\\_and\\_Practice\\_of\\_Online\\_Learning.pdf](http://biblioteca.ucv.cl/site/colecciones/manuales_u/99Z_Anderson_2008-Theory_and_Practice_of_Online_Learning.pdf) (accessed: 06.06.2017).

Andronico, A., Carbonaro, A., Casadei, G., Colazzo, L., Molinari, A., & Ronchetti, M. (2003). Integrating a multi-agent recommendation system into a mobile learning management system. *Proceedings of artificial intelligence in mobile system*, 123-132. Chicago

AOS (2015). Jack. Autonomous Decision Making System. <http://aosgrp.com/products/jack/> AUML-2 Tool. AUML-2 & Interaction Diagram Tool. <http://waitaki.otago.ac.nz/~michael/auml/> (Accessed 31<sup>st</sup> October 2016).

Austin, J. L. (1962). *How to do things with words*. Cambridge, MA: Harvard UP.

Bădică, C., Budimac, Z., Burkhard, H. D., & Ivanović, M. (2011). Software agents: Languages, tools, platforms. *Computer Science and Information Systems, ComSIS*, 8(2), 255-296.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P., (Editors). (2003). *The description logic handbook: theory, implementation, and applications*. Cambridge university press.

## References

- Baader, F., Horrocks, I., & Sattler, U. (2007). Description logics. *Foundations of Artificial Intelligence*, 3, 135-179.
- Baader, F., & Nutt, W. (2003, January). Basic description logics. In *Description logic handbook* (pp. 43-95).
- Baffes, P. T. (1994). Learning to model students: Using theory refinement to detect misconceptions. Technical Report, Artificial Intelligence, University of Texas at Austin.
- Baral, C., & Gelfond, M. (1994). Logic programming and knowledge representation. *The Journal of Logic Programming*, 19, 73-148.
- Bañeres, D. (2017). A Personalized Summative Model based on Learner's Effort. *International Journal of Emerging Technologies in Learning (iJET)*, 12(06), 4-21.
- Bañeres, D., & Conesa, J. (2017). A Life-long Learning Recommender System to Promote Employability. *International Journal of Emerging Technologies in Learning (iJET)*, 12(06), 77-93.
- Beginner SQL Tutorial (2017). Learn SQL Programming. <http://beginner-sql-tutorial.com/sql.htm> (Accessed: 2<sup>nd</sup> July 2017)
- Bellifemine, F., Poggi, A., & Rimassa, G. (1999, April). JADE—A FIPA-compliant agent framework. In *Proceedings of PAAM* (Vol. 99, No. 97-108, p. 33).
- Bellifemine, F. L., Caire, G., & Greenwood, D. (2007). *Developing multi-agent systems with JADE* John Wiley & Sons.
- Bench-Capon, T. J. (1998). Specification and implementation of toulmin dialogue game. *Proceedings of JURIX*, 98. pp. 5-20.
- Bordini, R. H., Hübner, J. F., & Tralamazza, D. M. (2006, May). Using Jason to implement a team of gold miners. In *International Workshop on Computational Logic in Multi-Agent Systems* (pp. 304-313). Springer Berlin Heidelberg.
- Bordini, R. H., Hübner, J. F., & Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using jason* John Wiley & Sons.
- Bratko, I. (2001). *Prolog programming for artificial intelligence* (3<sup>rd</sup> Ed.). Pearson education, England.
- Braubach, L., Pokahr, A., & Lamersdorf, W. (2004, September). Jadex: A short overview. In *Main Conference Net. ObjectDays* (Vol. 2004, pp. 195-207).
- Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3), 203-236.

## References

- Brewka, G., Eiter, T., & Truszczyński, M. (2011). Answer set programming at a glance. *Communications of the ACM*, 54(12), 92-103.
- Bull, J. & McKenna, C. (2004) *Blueprint for computer-assisted assessment*. RoutledgeFalmer, London.
- Casteel, C. (1988). Effects of chunked reading among learning disabled students: An experimental comparison of computer and traditional chunked passages. *Journal of Educational Technology Systems*, 17(2), 115-21.
- Castelfranchi, C. (1995). Commitments: From individual intentions to groups and organizations. *Icmas*, 95. pp. 41-48.
- Cernuzzi, L., & Zambonelli, F. (2009). Gaia4E: A Tool Supporting the Design of MAS using Gaia. In *ICEIS (4)* (pp. 82-88).
- Chadli, A., Bendella, F., & Tranvouez, E. (2015). A Two-Stage Multi-Agent Based Assessment Approach to Enhance Students' Learning Motivation through Negotiated Skills Assessment. *Journal of Educational Technology & Society*, 18(2), 140-152.
- Chin, K. O., Gan, K. S., Alfred, R., Anthony, P., & Lukose, D. (2014). Agent Architecture: An Overviews. *Transactions on science and technology*, 1(1), 18-35.
- Conole, G., & Warburton, B. (2005). A review of computer-assisted assessment. *Research in learning technology*, 13(1).
- Cossentino, M. (2005). From requirements to code with the PASSI methodology. *Agent-oriented methodologies*, 3690, 79-106.
- Cossentino, M., & Potts, C. (2002, June). A CASE tool supported methodology for the design of multi-agent systems. In *International Conference on Software Engineering Research and Practice (SERP'02)*.
- Cost, R. S., Chen, Y., Finin, T., Labrou, Y., & Peng, Y. (1999). Modeling agent conversations with colored petri nets. *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pp. 59-66.
- Da Silva, D. M., & Vieira, R. (2007). Argonaut: Integrating jason and jena for context aware computing based on owl ontologies. *Agent, Web Services, and Ontologies Integrated Methodologies (AWESOME'007)*, p.19.
- DAML (2006). *The DARPA Agent Markup Language Homepage*. <http://www.daml.org/> (accessed: November 17<sup>th</sup>, 2016).
- de Bruijn, J., Polleres, A., Fensel, D., & Motik, B. (2004). OWL lite. *WSML Deliverable D20 v 0.1*.

## References

- de Mantaras, R. L. (2001). Case-based reasoning. In *Machine Learning and Its Applications* (pp. 127-145). Springer Berlin Heidelberg.
- De Silva, L. (2009). Planning in BDI agent systems. PhD Thesis.
- Dekeyser, S., de Raadt, M., & Lee, T. Y. (2007, March). Computer assisted assessment of SQL query skills. In *Proceedings of the eighteenth conference on Australasian database-Volume 63* (pp. 53-62). Australian Computer Society, Inc..
- Dell'Acqua, P., Sadri, F., Toni, F., & Toni, F. S. F. (1999). *Communicating agents*. Citeseer.
- DeLoach, S. A. (2001). *Analysis and Design using MaSE and agentTool*. Air force inst of tech wright-patterson afb oh school of engineering and management.
- Dey, A. K., Abowd, G. D., & Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction, 16*(2), 97-166.
- Dogac, C., & Cingil, I. (2003). *B2B e-commerce technology: frameworks, standards and emerging issues*. Addison Wesley.
- DuCharme, B. (2013). *Learning Sparql*. O'Reilly Media, Inc. Beijing.
- Dutchuk, M., Mohammadi, K. A., & Lin, F. (2009). QuizMAster-A multi-agent game-style learning activity. In *Learning by playing. game-based education system design and development* (pp. 263-272). Springer Berlin Heidelberg.
- Ehimwenma, K. E., Beer, M. & Crowther, P. (Feb, 2016). *Computational Estimate Visualisation and Evaluation of Agent Classified Rules Learning System*. International Journal of Emerging Technologies in Learning (IJET). Vol.11(1). Pp. 38-47.
- Ehimwenma, K. E., Beer, M., & Crowther, P. (2015). *Student Modelling and Classification Rules Learning for Educational Resource Prediction in a Multiagent System*. 7th Computer Science and Electronic Engineering Conference (CEEC2015), IEEE. Pp. 59-64.
- Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2008). Combining answer set programming with description logics for the semantic web. *Artificial Intelligence, 172*(12), 1495-1539.
- El Mabrouk, M., Gaou, S., & Rtili, M. K. (2017). Towards an Intelligent Hybrid Recommendation System for E-Learning Platforms Using Data Mining. *International Journal of Emerging Technologies in Learning (iJET), 12*(06), 52-76.
- Finlay, J., & Dix, A. (1996). *An introduction to artificial intelligence*. Crc Press.

## References

FIPA (2000) (*Foundation for Intelligent and Physical Agents*). FIPA Ontology Service Specification. <http://www.fipa.org/>.

Gamalel-Din, S. (2002, June). The smart tutor: Student-centered case-based adaptive intelligent e-tutoring. In *the Proceedings of the 1st International Conference on Informatics and Systems, Cairo* (Vol. 17, p. 20).

Gelfond, M. (2008). Answer sets. *Foundations of Artificial Intelligence*, 3, 285-316.

Gelfond, M., & Lifschitz, V. (1988, August). The stable model semantics for logic programming. In *ICLP/SLP* (Vol. 88, pp. 1070-1080).

Genesereth, M. R., & Ketchpel, S. P. (1994). Software agents. *Commun.ACM*, 37(7), 48-53.

Gladun, A., Rogushina, J., Martínez-Béjar, R., & Fernández-Breis, J. T. (2009). An application of intelligent techniques and semantic web technologies in e-learning environments. *Expert Systems with Applications*, 36(2), 1922-1931.

GOAL. (2016) The GOAL Programming Language Home. <https://goalapl.atlassian.net/wiki/>

Goodwin, R. (1995). Formalizing properties of agents. *Journal of Logic and Computation*, 5(6), 763-781.

González, C., Burguillo, J. C., & Llamas, M. (2005). Case-Based student modeling in multi-agent learning environment. In *Multi-Agent Systems and Applications IV* (pp. 72-81). Springer Berlin Heidelberg.

Graesser, A. C., Hu, X., & McNamara, D. S. (2005). Computerized learning environments that incorporate research in discourse psychology, cognitive science, and computational linguistics. *Experimental Cognitive Psychology and its Applications: Festschrift in Honor of Lyle Bourne, Walter Kintsch, and Thomas Landauer*. Washington, DC: American Psychological Association.

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 43(5), 907-928.

Gruber, T. (1993). Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2), 199-220.

Hölldobler, S., & Schweizer, L. (2014, April). Answer Set Programming and CLASP A Tutorial. In *Young Scientists' International Workshop on Trends in Information Processing (YSIP)* (p. 77).

## References

Horridge, M., Knublauch, H., Rector, A., Stevens, R., & Wroe, C. (2004). A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0. *University of Manchester*.

Horrocks, I., Patel-Schneider, P. F., & Van Harmelen, F. (2003). From SHIQ and RDF to OWL: The making of a web ontology language. *Web semantics: science, services and agents on the World Wide Web*, 1(1), 7-26.

Hutchinson, A. (1994). *Algorithmic learning*. Oxford University Press, Inc.

Jena. An Introduction to RDF and Jena RDF API.

[https://jena.apache.org/tutorials/rdf\\_api.html](https://jena.apache.org/tutorials/rdf_api.html) (accessed: 27<sup>th</sup>, March 2017)

Jennings, N. R., & Wooldridge, M. (1995). Applying agent technology. *Applied Artificial Intelligence an International Journal*, 9(4), 357-369.

Jennings, N. R., Wooldridge, M., & Kinny, D. (1998). A methodology for agent-oriented analysis and design. In *Proc. 3rd Int Conference on Autonomous Agents*.

Jiang, H., & Huhns, M. N. (2005). *An approach to broaden the semantic coverage of ACL speech acts* (pp. 162-171). Springer Berlin Heidelberg.

Johnson, W. L., & Rickel, J. (1997). Steve: An animated pedagogical agent for procedural training in virtual environments. *ACM SIGART Bulletin*, 8(1-4), 16-21.

Kenny, C., & Pahl, C. (2005). *Automated tutoring for a database skills training environment* (Vol. 37, No. 1, pp. 58-62). ACM.

Klapiscak, T., & Bordini, R. H. (2009). JASDL: A practical programming approach combining agent and semantic web technologies. *Declarative agent languages and technologies VI* (pp. 91-110) Springer.

Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. *Emerging Artificial Intelligence Applications in Computer Engineering in I. Maglogianis et al (Eds.)*. IOS Press, pp. 3-24.

Labrou, Y., & Finin, T. (1998). Semantics and conversations for an agent communication language. *Readings in Agents*, 235-242.

Labrou, Y., & Finin, T. (1998). *Semantics for an agent communication language* (pp. 209-214). Springer Berlin Heidelberg.

Laclavik, M., Balogh, Z., Babik, M., & Hluchý, L. (2012). Agentowl: Semantic knowledge model and agent architecture. *Computing and Informatics*, 25(5), 421-439.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005, June). A study of the difficulties of novice programmers. In *ACM SIGCSE Bulletin* (Vol. 37, No. 3, pp. 14-18). ACM.

## References

- Laird, J. E. (2008). Extending the Soar cognitive architecture. *Frontiers in Artificial Intelligence and Applications*, 171, 224.
- Laird, J. E. & Congdon, B. C. (2015). The Soar User's Manual Version 9.5.0. <http://web.eecs.umich.edu/~soar/downloads/Documentation/SoarManual.pdf>
- Lans, R. F. (2006). Introduction to SQL: Mastering the relational database language. Addison-Wesley Professional
- Lifschitz, V. (2008, July). What Is Answer Set Programming?. In *AAAI* (Vol. 8, pp. 1594-1597).
- Logan, b. (2014). Designing Intelligent Agents [Lecture Slides]. <http://www.cs.nott.ac.uk/> (Accessed: November 11th, 2015.)
- Maedche, A., & Staab, S. (2001). Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16(2), 72-79.
- Maes, P. (1991). The agent network architecture (ANA). *Acm sigart bulletin*, 2(4), 115-120.
- Manouselis, N., Drachsler, H., Vuorikari, R., Hummel, H., & Koper, R. (2011). Recommender systems in technology enhanced learning. *Recommender systems handbook*, 387-415.
- Marsland, S. (2009, 2014). *Machine learning: an algorithmic perspective*. CRC press.
- Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.). (2013). *Machine learning: An artificial intelligence approach*. Springer Science & Business Media.
- Mitrovic, A. (1998): Learning SQL with a computerized tutor. In SIGCSE'98 pp. 307 - 311.
- Monette, D. (2014). Introduction to Agents and Multi-agent Systems [presentation slides]. Europe Week, 3<sup>rd</sup> – 7<sup>th</sup> March 2014, University of Hertfordshire, Hatfield. <http://www.monettdiaz.com/> (accessed: October 30<sup>th</sup>, 2015).
- Morandini, M., Nguyen, D. C., Penserini, L., Perini, A., & Susi, A. (2011). Tropos Modeling, Code Generation and Testing with the Taom4E Tool. In *iStar* (pp. 172-174).
- Nardi, D. & Brachman, R. J. An Introduction to Description Logics (Chapter 1). In F., Baader, D. L., McGuinness, D. Nardi & P. F., Patel-Schneider. (Ed.), (2003). *Description logic handbook*. (pp. 1-40).
- O'Reilly, M. & Morgan, C. (1999) Online assessment: creating communities and opportunities, in: S. Brown, P. Race & J. Bull (Eds) *Computer-assisted assessment in higher education* (London, Kogan Page), 149–161.

## References

- Obitko, M. (2007). Ontologies of the Semantic Web. <https://www.obitko.com/tutorials/ontologies-semantic-web/description-logics.html> (accessed: February, 19<sup>th</sup> 2017).
- Oishi, E. (2006). Austin's speech act theory and the speech situation. *Esercizi Filosofici*, 1(2006), 1-14.
- Padayachee, I. (2002). Intelligent tutoring systems: Architecture and characteristics. *University of Natal, Durban, Information Systems & Technology, School of Accounting & Finance*.
- Padgham, L and Singh, D 2013, 'Situational preferences for BDI plans', in Takayuki Ito, Catholijn Jonker, Maria Gini, Onn Shehory (ed.) Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), New York, NY, USA, 6-10 May 2013, pp. 1013-1020.
- Padgham, L., & Winikoff, M. (2005). *Developing intelligent agent systems: A practical guide* John Wiley & Sons.
- Padgham, L., Thangarajah, J., & Winikoff, M. (2008, July). Prometheus Design Tool. In *AAAI* (Vol. 8, pp. 1882-1883).
- Patterson, D. (1990). *Introduction to artificial intelligence and expert systems*. Prentice-Hall, Inc..
- Pavlov, I. P. (1960). *Conditioned reflexes : an investigation of the physiological activity of the cerebral cortex*. Oxford University Press, London.
- Peredo, R., Canales, A., Menchaca, A., & Peredo, I. (2011). Intelligent web-based education system for adaptive learning. *Expert Systems with Applications*, 38(12), 14690-14702.
- Pipitone, A., Cannella, V. & Pirrone, R. (2012). Cognitive Models and their Application in Intelligent Tutoring System (Chapter 2). In Gigliola Paviotti, Pier Giuseppe Rossi & Dens Zarka (Ed.). *Intelligent Tutoring Systems: An Overview*.
- Pitkäranta, T. (2004). *Software Agents in Semantic Web Environment*. Doctoral Dissertation, Helsinki University of Technology, Finland.
- Piunti, M., Ricci, A., Boissier, O., & Hübner, J. F. (2009). Embodied organisations in MAS environments. In *Multiagent System Technologies* (pp. 115-127). Springer Berlin Heidelberg.
- Prior, J. C. (2003, January). Online assessment of SQL query formulation skills. In *Proceedings of the fifth Australasian conference on Computing education-Volume 20* (pp. 247-256). Australian Computer Society, Inc..



## References

Prior, J. C., & Lister, R. (2004). The backwash effect on SQL skills grading. *ACM SIGCSE Bulletin*, 36(3), 32-36.

Python (2016). Enthought Canopy. <https://www.enthought.com/products/canopy/>. Enthought Inc.

Ricci, A., Piunti, M., & Viroli, M. (2011). Environment programming in multi-agent systems: An artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2), 158-192.

Ricci, A., Viroli, M., & Omicini, A. (2006). CArtAgO: An infrastructure for engineering computational environments in MAS. In D. Weyns, H.V.D. Parunak, & M. Fabien, editors. *Second International Workshop on Environments for Multi-Agent Systems*, volume 4389 of *Lecture Notes in Computer Science*, Utrecht, The Netherlands, 2006. Springer-Verlag.

Rifkin, R., & Klautau, A. (2004). In defense of one-vs-all classification. *Journal of machine learning research*, 5(Jan), 101-141.

Ritter, S., Anderson, J., Cytrynowicz, M. & Medvedeva, O. Authoring Content in the Published PAT Algebra Tutor. *Journal of Interactive Media in Education*, 98 (9) 8 Oct. 1998 [[www-jime.open.ac.uk/98/9](http://www-jime.open.ac.uk/98/9)]

Rosbottom, J., & Moulin, C. (1998). Using intelligent agents to change the delivery of education (poster). *ACM SIGCSE Bulletin*, , 30. (3) pp. 303.

Rossi, P. G. & Fedeli, L. (2012). Intelligent Tutoring Systems: a short History and New Challenges (*Chapter 1*). *Intelligent Tutoring Systems (ITS)*. In Gigliola Paviotti, Pier Giuseppe Rossi & Dens Zarka (Ed.). *Intelligent Tutoring Systems: An Overview*. Pp. 13-56.

Rudolph, S. (2011). Foundations of description logics. In *Reasoning Web. Semantic Technologies for the Web of Data* (pp. 76-136). Springer Berlin Heidelberg.

Russell, S. J., & Norvig, P. (2010). *Artificial intelligence: A novel approach*. Pearson Education, New Jersey.

Sadiq, S., Orłowska, M., Sadiq, W., & Lin, J. (2004, June). SQLator: an online SQL learning workbench. In *ACM SIGCSE Bulletin* (Vol. 36, No. 3, pp. 223-227). ACM.

Schiffer, S. R. (1972). *Meaning*. Clarendon Press, Oxford.

Searle, J. R. (1969). *Speech acts: An essay in the philosophy of language* Cambridge university press.

Shoham, Y. (1991, July). AGENT0: A Simple Agent Language and Its Interpreter. In *AAAI* (Vol. 91, p. 704).

## References

Skinner, B. F. (1938). *The behavior of organisms: An experimental analysis*. Appleton-Century. New York.

SQLCourse.com (2017). Interactive Online SQL Training. <http://www.sqlcourse.com/table.html> (Accessed: 2<sup>nd</sup> July 2017)

SQLzoo (March 2017). [https://sqlzoo.net/wiki/SELECT\\_Quiz](https://sqlzoo.net/wiki/SELECT_Quiz) (Accessed: 2<sup>nd</sup> July 2017)

STEM (2013). Pre-Assessment. <http://www.stemresources.com/index.php?id=51&> (Accessed January 08, 2014).

SurveyMonkey (2017). SurveyMonkey. <https://www.surveymonkey.co.uk/>

Verbert, K., Manouselis, N., Ochoa, X., Wolpers, M., Drachsler, H., Bosnic, I., & Duval, E. (2012). Context-aware recommender systems for learning: a survey and future challenges. *IEEE Transactions on Learning Technologies*, 5(4), 318-335.

Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes*. Harvard university press.

W3C (2004). OWL Web Ontology Language. <https://www.w3.org/TR/webont-req/> (accessed: November 17<sup>th</sup>, 2016).

W3C Recommendation. (Feb, 2004). RDF/XML Syntax Specification (Revised). <https://www.w3.org/TR/REC-rdf-syntax/#figure2> (Accessed: March 28th, 2016).

W3C (2014). Resource Description Framework RDF <https://www.w3.org/RDF/> (accessed: December, 22<sup>nd</sup> 2016).

w3Schools.com (2017). [https://www.w3schools.com/sql/sql\\_intro.asp](https://www.w3schools.com/sql/sql_intro.asp) (Accessed: 2<sup>nd</sup> July 2017).

Wang, F. (2014). POMDP Framework for Building an Intelligent Tutoring System. *Computer Supported Education (CSEDU2014)*. SCITEPRESS, pp. 233-240.

Wooldridge, M. (2002). *An introduction to multiagent systems. First Ed.* John Wiley & Sons.

Wooldridge, M. (2009). *An introduction to multiagent systems. Second Ed.* John Wiley & Sons.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge engineering review*, 10(2), 115-152.

Wooldridge, M., Jennings, N.R. and Kinny, D. (2000). The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and multi-agent systems*, 3(3), pp.285-312.

## *References*

Wu, X., Zeng, G., & Yang, G. (2008, October). A Novel Approach for Describing Goals with DLs in Intelligent Agents. In *2008 Fourth International Conference on Natural Computation* (Vol. 6, pp. 226-230). IEEE.

Yu, S., & Zhiping, L. (2008, December). Intelligent pedagogical agents for intelligent tutoring systems. In *Computer Science and Software Engineering, 2008 International Conference on* (Vol. 1, pp. 516-519). IEEE.

Zhang, T. I., Kendall, E., & Jiang, H. (2002). An agent-oriented software engineering methodology with application of information gathering systems for LCC. In *for LLC, Procs AOIS-2002*.

Zhang, Z., Thangarajah, J., & Padgham, L. (2008, May). Automated unit testing intelligent agents in PDT. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers* (pp. 1673-1674). International Foundation for Autonomous Agents and Multiagent Systems.

Zini, F., & Sterling, L. (1999, September). Designing Ontologies for Agents. In *APPIA-GULP-PRODE* (pp. 29-42).

# Appendix A

## A.1 Pre-assessment Data

This is the student skills data, recorded and stored by the agent *agModel* (student) in the Pre-assessment System. Appended to each data is the date and time of each pre-assessment exercise. The time between each event was analysed and used to plot the binary classification graph in Chapter 7.

```
//The INSERT desired concept data
```

```
desired_Concept("INSERT, date(2017-1-26), time(12-10-23)") [source(agSupport)].
```

```
quizSelectWhere("What query statement will return the player number and address of each player living in Stratford? HINT: order of address: STREET, HOUSENO, POSTCODE., date(2017-1-26), time(12-10-23)") [source(agSupport)].
```

```
responseToSelectWhere("SELECT PLAYERNO, STREET, HOUSENO, POSTCODE, date(2017-1-26), time(12-13-54)") [source(agSupport)].
```

```
failed("The student has NOT passed the SELECT...WHERE question., date(2017-1-26), time(12-13-54)") [source(agSupport)].
```

```
quizSelectAll("State the SQL query that will output all the data in TENNIS_TEAMS?, date(2017-1-26), time(12-13-54)") [source(agSupport)].
```

```
responseToSelectAll("SELECT PLAYERNO, STREET, HOUSENO, POSTCODE, date(2017-1-26), time(12-13-59)") [source(agSupport)].
```

```
failed("The student has NOT passed the SELECT_ALL question., date(2017-1-26), time(12-13-59)") [source(agSupport)].
```

## *Appendix A*

### **//The INSERT desired concept data**

```
desired_Concept("INSERT", date(2017-1-26), time(12-14-40)) [source(agSupport)].
```

```
quizSelectWhere("What query statement will return the player number and address of each player living in Stratford? HINT: order of address: STREET, HOUSENO, POSTCODE., date(2017-1-26), time(12-14-40)") [source(agSupport)].
```

```
responseToSelectWhere("SELECT PLAYERNO, STREET, HOUSENO, POSTCODE FROM TENNIS_PLAYERS WHERE TOWN = 'Stratford', date(2017-1-26), time(12-14-46)") [source(agSupport)].
```

```
passed("The student has passed the SELECT...WHERE question., date(2017-1-26), time(12-14-46)") [source(agSupport)].
```

```
quizSelectAll("State the SQL query that will output all the data in TENNIS_TEAMS?, date(2017-1-26), time(12-14-46)") [source(agSupport)].
```

```
responseToSelectAll("SELECT * FROM TENNIS_TEAMS, date(2017-1-26), time(12-15-24)") [source(agSupport)].
```

```
passed("The student has passed the SELECT_ALL question., date(2017-1-26), time(12-15-30)") [source(agSupport)].
```

### **//The DELETE desired concept data**

```
desired_Concept("DELETE", date(2017-1-26), time(12-17-38)) [source(agSupport)].
```

```
quizInsertSelect("Enter into the table: TENNIS_RECR_PLAYERS; the number, name, town, and telephone number of each non-competition player? HINT: INSERT and SELECT., date(2017-1-26), time(12-17-38)") [source(agSupport)].
```

```
responseToInsertSelect("SELECT * FROM TENNIS_RECR_PLAYERS, date(2017-1-26), time(12-22-18)") [source(agSupport)].
```

```
failed("The student has NOT passed the INSERT with SELECT question., date(2017-1-26), time(12-22-18)") [source(agSupport)].
```

## *Appendix A*

```
quizInsertValue("A new team has enrolled in the league. The third  
team will be captained by player 100, and will compete in the third  
division. Add the team to the database?, date(2017-1-26), time(12-  
22-18)") [source(agSupport)].
```

```
responseToInsertValue("INSERT , date(2017-1-26), time(12-22-  
37)") [source(agSupport)].  
failed("The student has NOT passed the INSERT with VALUE question.,  
date(2017-1-26), time(12-22-37)") [source(agSupport)].
```

```
desired_Concept("SELECT, date(2017-1-26), time(12-28-  
10)") [source(agSupport)].
```

***//The INSERT desired concept data***

```
desired_Concept("INSERT, date(2017-1-26), time(12-29-  
43)") [source(agSupport)].
```

```
quizSelectWhere("What query statement will return the player number  
and address of each player living in Stratford? HINT: order of  
address: STREET, HOUSENO, POSTCODE., date(2017-1-26), time(12-29-  
43)") [source(agSupport)].
```

```
responseToSelectWhere("SELECT PLAYERNO, STREET, HOUSENO, POSTCODE  
FROM TENNIS_PLAYERS WHERE TOWN = 'STRATFORD';, date(2017-1-26),  
time(12-32-4)") [source(agSupport)].
```

```
failed("The student has NOT passed the SELECT...WHERE question.,  
date(2017-1-26), time(12-32-4)") [source(agSupport)].
```

```
quizSelectAll("State the SQL query that will output all the data in  
TENNIS_TEAMS?, date(2017-1-26), time(12-32-4)") [source(agSupport)].
```

```
responseToSelectAll("SELECT PLAYERNO, STREET, HOUSENO, POSTCODE,  
date(2017-1-26), time(12-33-6)") [source(agSupport)].
```

```
failed("The student has NOT passed the SELECT...WHERE question.,  
date(2017-1-26), time(12-33-6)") [source(agSupport)].
```

## *Appendix A*

### **//The UNION desired concept data**

```
desired_Concept("UNION, date(2017-1-26),time(12-42-14)") [source(agSupport)].
```

```
quizFullOuterJoin("Give, for each player, the player number, the name and the penalties incurred by him or her; order the result by player number. (HINT: you need to use OUTER JOIN), date(2017-1-26), time(12-42-14)") [source(agSupport)].
```

```
responseToFullOuterJoin("SELECT P.PLAYERNO, P.NAME, PEN.AMOUNT, date(2017-1-26), time(12-59-10)") [source(agSupport)].
```

```
failed("The student has NOT passed the FULL_OUTER_JOIN question., date(2017-1-26), time(12-59-10)") [source(agSupport)].
```

```
quizInnerJoin("For each player born after June 1920, find the name and the penalty incurred by him or her? HINT: you need to use INNER JOIN, date(2017-1-26), time(12-59-10)") [source(agSupport)].
```

```
responseToInnerJoin("SELECT P.PLAYERNO, P.NAME, PEN.AMOUNT FROM TENNIS_PLAYERS P INNER JOIN TENNIS_PENALTIES PEN ON P.PLAYERNO = PEN.PLAYERNO, date(2017-1-26), time(13-1-19)") [source(agSupport)].
```

```
failed("The student has NOT passed the INNER_JOIN question., date(2017-1-26), time(13-1-19)") [source(agSupport)].
```

### **//The JOIN desired concept data (SECOND ATTEMPT KEN)**

```
desired_Concept("JOIN, date(2015-10-16), time(11-0-15)") [source(agSupport)].
```

```
quizUpdateSelect("Set the number of sets won to zero for all players resident in Stratford., date(2015-10-16), time(11-0-15)") [source(agSupport)].
```

```
responseToUpdateSelect("SELECT * FROM TENNIS_MATCHES, date(2015-10-16), time(11-3-16)") [source(agSupport)].
```

## *Appendix A*

```
failed("The student has NOT passed UPDATE with SELECT question.,  
date(2015-10-16), time(11-3-16)") [source(agSupport)].
```

```
quizUpdateWhere("Change the value F in the SEX column of the PLAYERS  
table to W (women)., date(2015-10-16), time(11-3-  
16)") [source(agSupport)].
```

```
responseToUpdateWhere("UPDATE SEX FROM P WHERE SEX = 'F' TO SEX =  
'W', date(2015-10-16), time(11-5-1)") [source(agSupport)].
```

```
failed("The student has NOT passed UPDATE with WHERE question.,  
date(2015-10-16), time(11-5-1)") [source(agSupport)].
```

### ***//The INSERT desired concept data***

```
desired_Concept("INSERT, date(2015-10-16), time(11-11-  
47)") [source(agSupport)].
```

```
quizSelectWhere("What query statement will return the player number  
and address of each player living in Stratford? HINT: order of  
address: STREET, HOUSENO, POSTCODE., date(2015-10-16), time(11-11-  
47)") [source(agSupport)].
```

```
responseToSelectWhere("SELECT STREET, HOUSENO, POSTCODE FROM  
TENNIS_PLAYERS WHERE TOWN='Stratford';, date(2015-10-16), time(11-  
12-57)") [source(agSupport)].
```

```
passed("The student has passed the SELECT...WHERE question.,  
date(2015-10-16), time(11-12-57)") [source(agSupport)].
```

```
quizSelectAll("State the SQL query that will output all the data in  
TENNIS_TEAMS?, date(2015-10-16), time(11-12-  
57)") [source(agSupport)].
```

```
responseToSelectAll("SELECT * FROM TENNIS_TEAMS;, date(2015-10-16),  
time(11-13-51)") [source(agSupport)].
```

```
passed("The student has passed the SELECT_ALL question., date(2015-  
10-16), time(11-13-51)") [source(agSupport)].
```



## *Appendix A*

***//Other data are with no response from the student:***

```
desired_Concept("INSERT, date(2015-10-16), time(11-8-32)") [source(agSupport)].
```

```
quizSelectWhere("What query statement will return the player number and address of each player living in Stratford? HINT: order of address: STREET, HOUSENO, POSTCODE., date(2015-10-16), time(11-8-32)") [source(agSupport)].
```

***//Another data, also with no response from the student:***

```
desired_Concept("UPDATE, date(2015-10-16), time(11-7-10)") [source(agSupport)].
```

```
quizDeleteSelect("Delete all penalties who live in the same town as player 44, but keep the data for player 44., date(2015-10-16), time(11-7-10)") [source(agSupport)].
```

***//The UPDATE desired concept data***

```
desired_Concept("UPDATE, date(2015-3-7), time(11-3-17)") [source(agSupport)].
```

```
desired_Concept("UPDATE, date(2015-3-7), time(11-8-4)") [source(agSupport)].
```

```
quizDeleteSelect("Delete all penalties who live in the same town as player 44, but keep the data for player 44., date(2015-3-7), time(11-8-54)") [source(agSupport)].
```

```
responseToDeleteSelect("DELETE FROM (SELECT * FROM TENNIS_PENALTIES WHERE PLAYERNO = 44), date(2015-3-7), time(11-9-27)") [source(agSupport)].
```

```
failed("The student has NOT passed the DELETE with SELECT question.") [source(agSupport)].
```

## *Appendix A*

```
quizDeleteWhere("Delete all penalties incurred by player 44 in  
1980., date(2015-3-7), time(11-9-27)") [source(agSupport)].
```

```
responseToDeleteWhere("DELETE FROM SELECT * FROM TENNIS_PENALTIES  
WHERE PLAYERNO = 44, date(2015-3-7), time(11-12-  
10)") [source(agSupport)].
```

```
failed("The student has NOT passed the DELETE with WHERE  
question.") [source(agSupport)].
```

### ***//The UPDATE desired concept data***

```
desired_Concept("UPDATE, date(2015-5-7), time(11-11-  
31)") [source(agSupport)].
```

```
quizDeleteSelect("Delete all penalties who live in the same town as  
player 44, but keep the data for player 44., date(2015-5-7),  
time(11-11-31)") [source(agSupport)].
```

```
responseToDeleteSelect("DELETE FROM TENNIS_PENALTIES(SELECT * FROM  
TENNIS_PENALTIES WHERE PLAYERNO = 44), date(2015-5-7), time(11-12-  
10)") [source(agSupport)].
```

```
failed("The student has NOT passed the DELETE with SELECT  
question.") [source(agSupport)].
```

```
quizDeleteWhere("Delete all penalties incurred by player 44 in  
1980., date(2015-5-7), time(11-12-10)") [source(agSupport)].
```

```
responseToDeleteWhere("DELETE * FROM TENNIS_PENALTIES WHERE PLAYERNO  
= 44 AND PAYMENT_DATE LIKE '1980', date(2015-5-7), time(11-14-  
4)") [source(agSupport)].
```

```
failed("The student has NOT passed the DELETE with SELECT  
question.") [source(agSupport)].
```

## *Appendix A*

### **//The UNION desired concept data**

```
desired_Concept("UNION, date(2015-3-7),time(11-19-4)") [source(agSupport)].
```

### **//Re-entering of desired\_Concept after studying quiz and database**

```
desired_Concept("UNION, date(2015-3-7),time(11-28-48)") [source(agSupport)].
```

```
quizFullOuterJoin("Give, for each player, the player number, the name and the penalties incurred by him or her; order the result by player number. (HINT: you need to use OUTER JOIN), date(2015-3-7), time(11-28-48)") [source(agSupport)].
```

```
responseToFullOuterJoin(SELECT * FROM TENNIS_PLAYERS (alias) P OUTER JOIN TENNIS_PENALTIES (alias) PEN ON P.PLAYERNO = PEN.PLAYERNO, date(2015-3-7), time(11-28-56)") [source(agSupport)].
```

```
failed("The student has NOT passed the FULL_OUTER_JOIN question.") [source(agSupport)].
```

```
quizInnerJoin("For each player born after June 1920, find the name and the penalty incurred by him or her? HINT: you need to use INNER JOIN, date(2015-3-7), time(11-28-56)") [source(agSupport)].
```

```
responseToInnerJoin(SELECT * FROM TENNIS_PLAYERS (alias) P INNER JOIN TENNIS_PENALTIES (alias) PEN ON P.PLAYERNO = PEN.PLAYERNO, date(2015-3-7), time(11-29-35)") [source(agSupport)].
```

```
failed("The student has NOT passed the INNER_JOIN question.") [source(agSupport)].
```

### **//The UNION desired concept data (SECOND ATTEMPT KEN)**

```
desired_Concept("UNION, date(2015-3-7),time(11-29-48)") [source(agSupport)].
```

```
quizFullOuterJoin("Give, for each player, the player number, the name and the penalties incurred by him or her; order the result by
```

## *Appendix A*

```
player number. (HINT: you need to use OUTER JOIN), date(2015-3-7),  
time(11-29-48)") [source(agSupport)].
```

```
responseToFullOuterJoin(SELECT P.PLAYERNO, P.NAME,  
PEN.PLAYERNO FROM TENNIS_PLAYERS (alias) P OUTER JOIN  
TENNIS_PENALTIES (alias) PEN ON P.PLAYERNO = PEN.PLAYERNO,  
date(2015-3-7), time(11-31-43)") [source(agSupport)].
```

```
failed("The student has NOT passed the FULL_OUTER_JOIN  
question.") [source(agSupport)].
```

```
quizInnerJoin("For each player born after June 1920, find the name  
and the penalty incurred by him or her? HINT: you need to use INNER  
JOIN, date(2015-3-7), time(11-31-43)") [source(agSupport)].
```

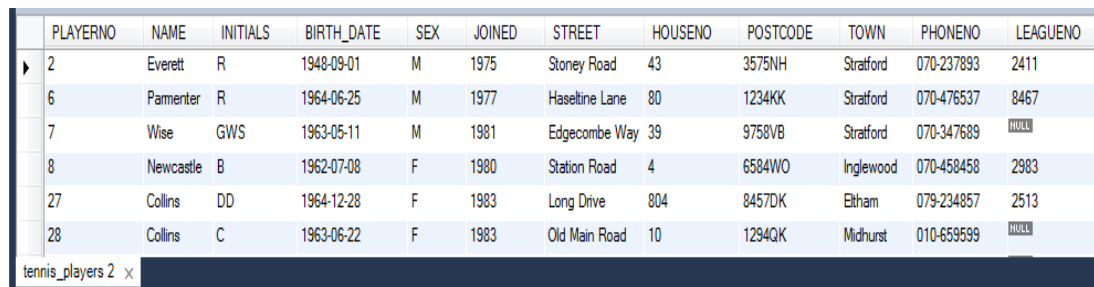
```
responseToInnerJoin(SELECT P.PLAYERNO, P.NAME,  
PEN.PLAYERNO FROM TENNIS_PLAYERS (alias) P INNER JOIN  
TENNIS_PENALTIES (alias) PEN ON P.PLAYERNO = PEN.PLAYERNO,  
date(2015-3-7), time(11-34-04)") [source(agSupport)].
```

```
failed("The student has NOT passed the INNER_JOIN  
question.") [source(agSupport)].
```

## Appendix A

### A.2 The MySQL Tennis\_Database Tables

The Tennis Database tables in the MySQL database that students used during their pre-assessment sessions.



PLAYERNO	NAME	INITIALS	BIRTH_DATE	SEX	JOINED	STREET	HOUSENO	POSTCODE	TOWN	PHONENO	LEAGUENO
2	Everett	R	1948-09-01	M	1975	Stoney Road	43	3575NH	Stratford	070-237893	2411
6	Pamenter	R	1964-06-25	M	1977	Haseltine Lane	80	1234KK	Stratford	070-476537	8467
7	Wise	GWS	1963-05-11	M	1981	Edgecombe Way	39	9758VB	Stratford	070-347689	NULL
8	Newcastle	B	1962-07-08	F	1980	Station Road	4	6584WO	Inglewood	070-458458	2983
27	Collins	DD	1964-12-28	F	1983	Long Drive	804	8457DK	Eltham	079-234857	2513
28	Collins	C	1963-06-22	F	1983	Old Main Road	10	1294QK	Midhurst	010-659599	NULL

Fig. 1: Snapshot of The Tennis\_Players Table



TEAMNO	PLAYERNO	DIVISION
1	6	first
2	27	second
4	100	fifth
5	100	sixth
*	NULL	NULL

Fig. 2: The Tennis\_Teams Table



PAYMENTNO	PLAYERNO	PAYMENT_DATE	AMOUNT
1	6	1980-12-08	100.00
2	44	1981-05-05	75.00
3	27	1983-09-10	100.00
4	104	1984-12-08	50.00
5	44	1980-12-08	25.00
6	8	1980-12-08	25.00

Fig. 3: The Tennis\_Penalties Table

Appendix A

	MATCHNO	TEAMNO	PLAYERNO	WON	LOST
▶	1	1	6	3	1
	2	1	6	2	3
	3	1	6	3	0
	4	1	44	3	2
	5	1	83	0	3
	6	1	2	1	3

tennis\_matches 5 ×

Fig. 4: The Tennis\_Matches Table

	PLAYERNO	BEGIN_DATE	END_DATE	POSITION
▶	2	1990-01-01	1992-12-31	Chairman
	2	1994-01-01	NULL	General member
	6	1990-01-01	1990-12-31	Secretary
	6	1991-01-01	1992-12-31	General member
	6	1992-01-01	1993-12-31	Treasurer
	6	1993-01-01	NULL	Chairman

tennis\_committee\_members 4 ×

Fig. 5: The Tennis\_Committee\_Members Table

	PLAYERNO	NAME	TOWN	PHONENO
▶	7	Wise	Stratford	070-347689
	28	Collins	Midhurst	010-659599
	39	Bishop	Stratford	070-393435
	95	Miller	Douglas	070-867564
*	NULL	NULL	NULL	NULL

tennis\_recr\_players 6 ×

Fig. 6: The Tennis\_Recr\_Players Table

# Appendix B

## B.1 Students' Feedback Questionnaire

### Evaluation of SQL Based Multiagent Pre-assessment System

1. Course of study

2. Year of study

3. The system was useful

- Strongly agreed
- Agreed
- Undecided
- Disagreed
- Strongly disagreed

4. The system helped to recall my previous knowledge

- Strongly agreed
- Agreed
- Undecided
- Disagreed
- Strongly disagreed

5. The system supports the learning of SQL

- Strongly agreed
- Agreed
- Undecided
- Disagreed
- Strongly disagreed

**6. I am not familiar with SQL**

- Strongly agreed
- Agreed
- Undecided
- Disagreed
- Strongly disagreed

**7. The system provided guidance to learning materials**

- Strongly agreed
- Agreed
- Undecided
- Disagreed
- Strongly disagreed

**8. The system has a use-able interface**

- Strongly agreed
- Agreed
- Undecided
- Disagreed
- Strongly disagreed

**9. I understood the purpose of the system**

- Strongly agreed
- Agreed
- Undecided
- Disagreed
- Strongly disagreed



*Appendix B*

10. The tutor was helpful in introducing the system

- Strongly agreed
- Agreed
- Undecided
- Disagreed
- Strongly disagreed

11. The tutor was helpful in providing assistance

- Strongly agreed
- Agreed
- Undecided
- Disagreed
- Strongly disagreed

12. The session's organisation was a good learning experience

- Strongly agreed
- Agreed
- Undecided
- Disagreed
- Strongly disagreed

13. The session was well organised

- Strongly agreed
- Agreed
- Undecided
- Disagreed
- Strongly disagreed

14. What was most interesting about the session's organisation?

*Appendix B*

15. What was least interesting about the session's organisation?

16. What most interesting about the SQL system?

17. What was least interesting about the SQL system?

## B.2 Consent Form

### Introduction to SQL: Evaluation of SQL Based Multiagent Pre-assessment System

#### Your Consent:

This session is about the evaluation of a system we are designing. The learning content on this system is SQL: structured query language. The system is to check whether a student is ready to learn the topic he/she desires to learn. This readiness is checked by first asking you questions on the next immediate-lower topic to the one you would enter. Each topic has two questions. If the answers you provide are correct, you will learn the topic you have entered. But if both answers are incorrect, you will be required to learn both. And if one is answered correctly and the other incorrectly, the incorrectly answered will be the one to be learnt.

We kindly request that you help to participate in this system's test and survey. Your response are anonymous and will be used to improve the design, content and performance of this system. Your consent and participation is significant to us. We won't take much of your time.

**NB:** Please, kindly complete the questionnaire when you finish with the system.

Thank you.

#### Objectives of the System:

Are to:

- 1) identify whether you are ready to learn the SQL topic you entered;
- 2) ensure that you have mastered an immediate-lower topic before learning a higher one;
- 3) direct you to the appropriate URL link that you can place on a browser.

**I agree to participate (a tick please):**    Yes     No

**Email:** .....

**Sign:**.....

## **B.3 Research Ethics Approval**

Howson, Tracey D <T.D.Howson@shu.ac.uk>

To

Ehimwenma, Kennedy K (student - 55002)

CC

Crowther, Paul

Today at 10:30 AM

Hi Ken

Please see the message below from the Ethics Committee Chair regarding your SHUREC1, please keep this safe.

*He seems to be researching other computing students on learning in computing so will not need a SHUREC2A and so does not need formal ethical approval. However, please would you feed back to him that he needs to make sure that he gives each of his research participants an information sheet telling them about the research and gets them to sign a consent form to ensure they have consented to the research. He needs to offer participants the chance to withdraw from the research at any time up to the submission of his thesis. He should also confirm that participants' data is anonymised and kept securely. He should send in a copy of his consent/information sheet and we will file it with his SHUREC1.*

Kind regards

*Tracey Howson*

Admin Officer

Cultural, Communication & Computing Research Institute (C3RI)

9104 Cantor Building, 153 Arundel Street, Sheffield, S1 2NU

Tel +44 (0)114 225 6741

Fax +44 (0)114 225 6702

Email [t.d.howson@shu.ac.uk](mailto:t.d.howson@shu.ac.uk)

Web <http://www.shu.ac.uk/research/c3ri/>

My web profile <http://www.shu.ac.uk/research/c3ri/people/tracey-howson>

## **B.4 Certificate of Volunteer Participants in the Survey**



### **Certificate of Participation**

This is to certify that

Xxxxx Xxxxx

participated in a Doctor of Philosophy research project testing a system to determine pre knowledge of SQL

Date January 2017

Dr Paul Crowther  
Deputy Head of the Department of Computing  
Sheffield Hallam University

A handwritten signature in black ink, appearing to read "P. Crowther", written over a light grey rectangular background.