APPLICATION OF MACHINE LEARNING FOR EXTRACTING PROGRAMMING LANGUAGE CONSTRUCTS FROM 4GL LEGACY CODE

W. S. A. Ilakshini C. Subasinghe

138237A

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

May 2015

APPLICATION OF MACHINE LEARNING FOR EXTRACTING PROGRAMMING LANGUAGE CONSTRUCTS FROM 4GL LEGACY CODE

W. S. A. Ilakshini C. Subasinghe

138237A

Dissertation submitted in partial fulfilment of the requirements for the degree Master of Science in Computer Science specializing in Software Architecture

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

May 2015

DECLARATION OF THE CANDIDATE & SUPERVISOR

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis/dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: ..... Date: .....

Name: W. S. A. Ilakshini C. Subasinghe

The above candidate has carried out research for the Masters Dissertation under my supervision.

Signature: ..... Date: .....

Name of Supervisor: Dr. Amal Shehan Perera

ACKNOWLEDGEMENTS

I most gratefully acknowledge my appreciation to Dr. Shehan Perera for his acceptance, guidance and encouragement as my supervisor for this research, without which this work would not have been successful. I further extend my appreciation to Dr. Malaka Walpola and all my lecturers at the department for their encouragement and support to complete the research accurately in a timely manner.

I sincerely acknowledge the support my colleagues and seniors at work have provided, especially Mr. Mifraz Marzoon for his invaluable expertise as well as my friends who motivated me to complete the research.

Last but not least, I extend my deepest gratitude towards my parents and family for their tolerance and undivided love and support throughout the program.

ABSTRACT

With the progression and innovations of the Information Technology industry, computer systems have become not only a part of an organization but the heart of it that drives their daily routines and manages and tracks the entire business process for most enterprises and for decades Advanced Business Languages (ABL) have been evolving to provide successful economic solutions to drive these businesses. Progress 4GL (Fourth Generation Language) is one such Advanced Business Language where organizations have developed entire business process on for 30 years. However, with the advancement of Free and Open Sourced Software providing business solutions, some organizations using these legacy systems are looking for means of migration. Even though proprietary service providers exists for the migration process, organizations with decades old data are reluctant to use them for both cost and security reasons. Yet, in house development is also costly since ABL experts are very few and would require much time and effort to complete the process.

This research project is focused on a solution to develop such expert system that can interpret progress 4GL code to aid not only enterprises with migration but also engineers to learn and understand the language logic with ease. With the use of the Machine Learning technologies where research concerning modelling human thinking into machines are popular, this thesis provides a Proof of Concept for a methodology in which, an expert system can be created to read 4GL code, analyse the code, understand and infer the code logic and output the workflow in a graphical Flow Chart format. The prototype is run through several training 4GL programs to evaluate the implementation of the proposed theory. Current application proves to be successful for code with simple syntax and leaves room for further improvements to the system that can be enhanced to process 4GL's many complex and evolving constructs and also the possibility of translating to a different language.

Keywords: Expert Systems, Natural Language Processing, CLIPSJNI, Progress 4GL, mxGraph, Java-ML, Proparse

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
4GL	Fourth Generation Languages
ABL	Advanced Business Language
AI	Artificial Intelligence
API	Application Program Interface
ANTLR	Another Tool for Language Recognition
CHUI	Character User Interface
CLIPS	C Language Integrated Production System – Expert System Dev Tool
CRUD	Create, Read, Update and Delete Operations
EBMT	Example Based Machine Translation
EGL	Enterprise Generation Language
GUI	Graphical User Interface
NLP	Natural Language Processing
PSC	Progress Software Corporation
RBMT	Rule Based Machine Translation
SDL	Specification and Description Language
WEKA	Weikato Environment for Knowledge Analysis

LIST OF APPENDICES

1. INTRODUCTION

Since its introduction to the corporate world decades ago Information Technology has become a driving force in the business world allowing corporations to not only broaden their spectrum of work but also to carry out their daily business functions; and Fourth Generation Languages (4GL) have been a popular option in developing customized application Software for these businesses. Progress Software Corporation's OpenEdge Advanced Business Language known commonly as Progress 4GL is one such 4GL that has been helping organizations create and run the best business applications since its first commercial release in 1984 [1-4]. Even though they provide quality economic services, some organizations using these legacy 4GL systems are looking for migration options to free and open source platforms since migration seems more commendable than the cost of upgrading and maintaining their current proprietary environments.

However, there is a certain risk in migrating to a new environment due to the organization's massive domain knowledge which is captured in the billions of lines of code in the program files written in 4GL. In addition, the lack of 4GL programming knowledge experts makes the task more tedious and impractical.

There are many number of proprietary software available to translate 4GL code to other open source languages such as JAVA but using such software puts the organization at the risk of sharing sensitive business information to a third party source. This particular problem has been addressed by many researches giving solutions from reverse engineer 4GL systems to methodical step by step translation of legacy systems. However, none provide a common solution that can be applied to any 4GL language legacy systems and to be used without fearing any sensitive data leaks.

Therefore, this research proposes one methodology that can be helpful for organizations looking for systems migration without having to use any additional expertise and resources while protecting the organizational data and finances. In the technological work where research concerning modelling human thinking into machines are popular, a system that is capable of learning a programming language to describe the workflow or even to translate the code would be ideal to solve the problem at hand. Therefore, the research is a proof of concept of extracting out the programming language constructs from the 4GL code approach with use of machine

learning technology and logical programming in order to support organizations to shift platforms.

1.1. Thesis Statement

The aim of the thesis is to identify a methodology where a computer system is capable of interpreting a program written in 4GL code and by doing so aid organizations with a practical solution for code migration to a new environment. The thesis looks into various 4GL interpretation and reverse engineering technologies as well as machine learning technologies that can be used to define the rule base and inference engine to interpret the 4GL code. The goal of the thesis is to provide a proof of concept to implement such intelligent system capable of learning a 4GL programming language. Even though research on conversion of programming language from one natural language to another (English to Spanish) is available and conversion between various other programming languages exists, research on training a machine to learn or identify a programming language is limited. Therefore, this thesis further aims at contributing as a stepping stone to the idea of Programming Language Processing taking Natural Language Processing as the baseline.

1.2. Thesis Overview

This thesis consists of five chapters structured as follows:

- **Chapter 2: Background**

  This chapter provides a detailed review of 4GL languages, 4GL reverse engineering concepts, Machine Learning and as well as Natural Language Processing concepts that forms the foundation for the research.

- **Chapter 3: Application Of Machine Learning For Extracting Programming Language Constructs From 4GL Legacy Code**

  The concepts and construction of the methodology in achieving the thesis goal are detailed out in this chapter. The process of achieving the goals is also explained is also described here.

- **Chapter 4: System Implementation & Evaluation**

This chapter contains the details of the implementation, integration and the validation process of the proposed system. The chapter also highlights the results and the limitations faced when implementing the proposed solution.

- **Chapter 5: Conclusions & Recommendations**

  The final chapter contains the description of the current status of the research project and the summarization of the achievements and the future work as well as discussion on the recommendations to continue and improve the current research.

2. BACKGROUND

The research area is concerned about creating a system that is intelligent to learn Progress® 4GL and interpret the workflow of an input of Progress® code. There are several areas of interest when looking into existing work that is related to the research and can be categorized overall as, research related any kind of 4GL programming language; any existing applications systems or alternate approaches available on solving the business problem; research related to building any kind of translators for programming languages that can understand or interpret coding and finally research related to building expert AI systems. With the findings of these key areas, the methodology on arriving at a solution can be outlined. The findings are important not only to understand the domain on which it is used, but also to integrate different existing techniques to successfully create the machine learning application for extracting 4GL Programming Language constructs. Below sections explains the findings for each category.

2.1. 4GL Application Environments

Fourth Generation Programming Languages and environments are very high level programming languages that are built for a specific purpose and are more oriented towards problem solving or development of commercial business software [5].There are many such 4GL languages available and are categorized depending on the type of service they focus on. SQL, Python, Informix 4GL, Ingres 4GL and Progress 4GL are some popular examples for 4GL languages. SQL and Python are generic 4GL languages and Ingres 4GL is a Database Query Language while Progress® 4GL and Informix 4GL provide Data manipulation, analysis, reporting functionalities to GUI Creation services to database-driver GUI application development functionalities. One thing that the latter three 4GL languages have in common is the fact that they all lack expertise on domain knowledge in companies using the technology and the cost of maintaining a legacy system running on these languages therefore is high.

IBM® Informix® 4GL is a comprehensive fourth-generation application development and production environment that provides users with flexibility without

the need for third-generation languages like C and COBOL [6]. It provides tools to create sophisticated database application with a consistent interface and with the capability to provide high-performance application execution. Informix 4GL allow easy deployment of functions on the Web and leverages any existing 4GL code to create Web services for applications that can be written in other programming languages such as Java and .Net without having to write and actual Java or .Net code.

Similarly, Progress® 4GL which was founded in 1981 and first publicly traded on NASDAQ in 1991 has more than 30 years of experience in application development and deployment and it helps organizations to create and run business application to best suit their needs [7]. They are focused on building the next generation of Cloud computing and are motivated to putting world-class, real-time analytics, decision-making, and data visualization at the fingertips of business users making the development, deployment, and management of business applications even simpler on any platform, any device, and any Cloud. Progress Software company statistics states that their products are purchased by more than 140,000 enterprises in 175+ countries and more than 4 million people are using applications built on their Progress technology [7].

Progress® OpenEdge® provides a modern, future-proof development platform for building dynamic, multi-tenant, multi-language applications across any platform, any mobile device, any Cloud as illustrated by Figure 2.1. It helps businesses to react in an instant to evolving market conditions, customer demands, regulatory changes, and the competitive landscape [7].

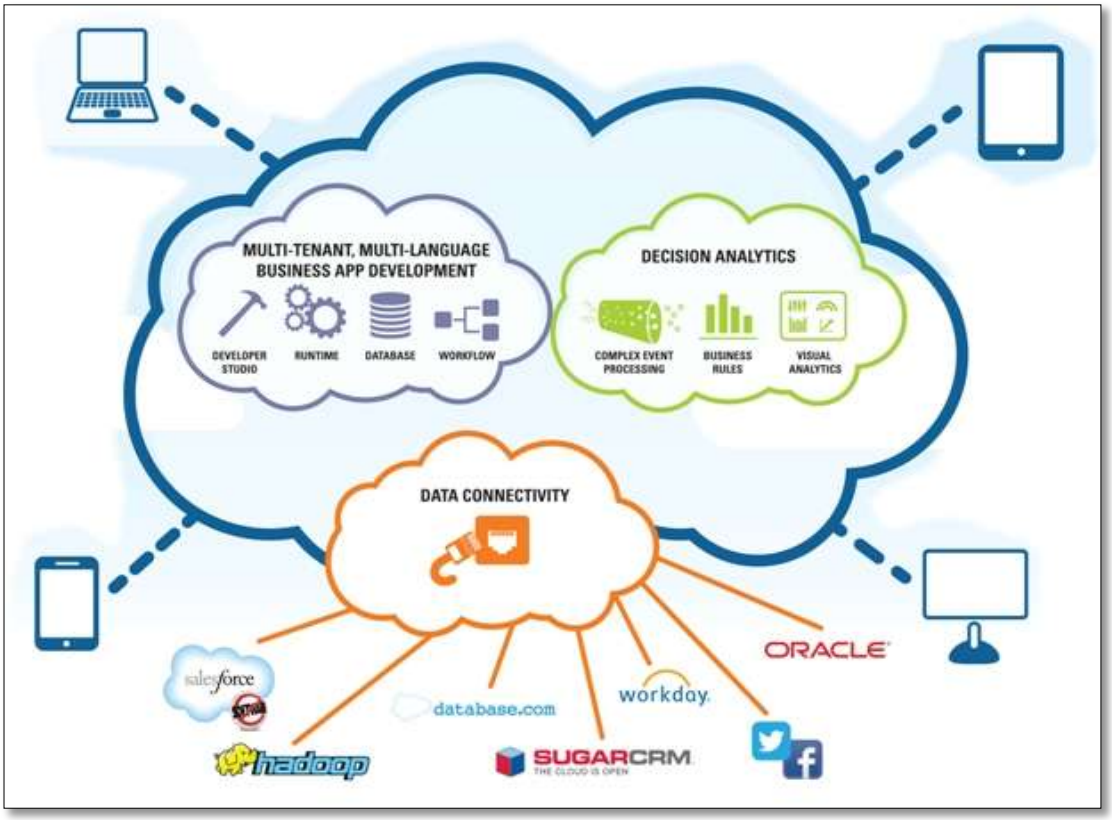


Figure 2.1: Next-generation application development and deployment: Source [7]

However, even with the promising future of these 4GL software application environments, not all enterprises are capable of continuing to grow with this proprietary software. A common problem statements highlighted in most of the research carried out on 4GL systems are that of its Cost on the licensing as well as the special technology and domain knowledge that is required to continue the business. Some such researches are [9], [10], [11] and [12]. Therefore, some enterprises built on Progress® who are pressured to continue to lookout for specialized resources and training as well as to bear the cost of upgrades and licensing are now looking into feasible opportunities to migrate to different platforms which would not disrupt their business process.

2.2. 4GL Conversion and Migration

There are many systems that have been developed which supports 4GL conversion and migration to various different languages, databases environments or platforms. IBM themselves have provided several transition and migration support

for its Informix 4GL to support their users for better transformation. Some of these tools and software are described below with reference to [8] and [13] - [21].

- IBM Informix 4GL to EGL Migration

Informix 4GL of IBM is designed for a specific purpose and it is ideally suited for its purpose although, since Informix 4GL only supports character based end user programs (CHUI) rather than graphical end-user programs (GUI) and since it only operates against Informix databases, some are looking into transitioning from Informix 4GL to EGL (Enterprise Generation Language) [8]. IBM EGL is a development environment and a structured programming language used for development of business application which outputs JAVA/J2SE or JAVA/J2EE code as required for the application. The transition support provided is for Informix 4GL only. If not businesses should build up their system from EGL.

- IBM Informix to Oracle Migration

SQLWays migration is a tool that is capable of automatically converting data, stored procedures, functions, triggers, database schema (DDL), and other database objects such as indexes and views and is recommended for Informix to Oracle Migration [13]. The software helps convert Informix server-side logic as well as front-end queries and scripts to confirm Oracle SQL syntax.

- IBM Informix 4GL Applications to Informix Genero

Informix Genero is another specialized and modernized 4GL application development and deployment environment that supports rapid application development and supports creating applications for Service Oriented Architecture (SOA) [14]. More importantly it is focused on reducing cost of re-development and protecting the organizations domain knowledge and the investment made on Informix 4GL applications [14], [15].

- 4GL Connector

The MoreData Company provides specialized services for Informix 4GL such as Performance Tuning, Load Balancing, Replication Setup, Backup Setup etc. The company further provides an Open Source technology solution called the 4GL

connector that would expose 4GL functions as EJB3 or Web services bypassing integration issues caused and allowing 4GL code to be used in SOA / Java EE systems without any code conversion [16], [17].

- FreeSoft

FreeSoft is a technology service provider that has been involved in technology migration since 1998. It provides a series of automated conversion technologies, project methodologies and processes to understand and transform a variety of legacy systems into modern IT environment and new architecture, while ensuring the lowest risk and cost [18]. Their legacy modernization services enable customers to transform different legacy systems running on heterogeneous proprietary platforms, developed in the listed programming languages with data stored in their legacy databases, into a Java EE or .NET technology platform. However, with using such third party vendor, businesses need to share their confidential domain knowledge and work as well as to bear the cost of services provided.

- 4GL Language Conversion to SAS Software

The paper Conversion to SAS Software from other 4GL languages [19] provides three primary reasons as to why the conversion is needed. First two reasons of which are the most common in all 4GL conversions: Cost and the focus skill set and training. The third reason stated is strategic issues. The strategic issues are further categorized as Platform Crossover, Legacy Systems and Long Term Viability. In consideration with the business problem in focus, all three of the issues are of major concern. The paper further gives distinct guidelines as to how the conversion should be carried out from identifying active programs, identifying the data used to deciding whether to mimic the existing system or improve the system.

- Progress 4GL to Java Conversions

There are conversion and migration tools available for Progress® 4GL as well [20], [21]. These tools are proprietary and would cost the business not only financially but also with the risk of exposing sensitive domain knowledge to a third party vendor. One such tool is the RainCode software that translates the procedural

Progress 4GL in to object oriented Java code operating on either Oracle or SQL database to a similar environment as the existing system or a new thin-client server based deployment architecture as shown in Figure 2.2. [20].

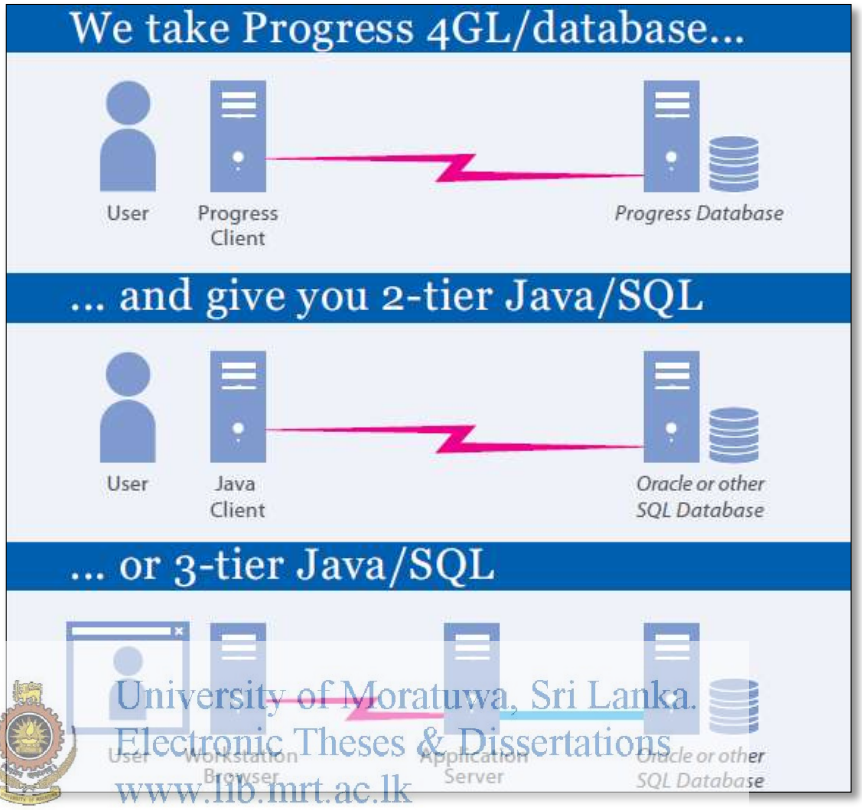


Figure 2.2: Automatic migration of Progress 4GL application to Java: Source [20]

Similar software is the Golden Code Development Corporation's Progress 4GL to Java Conversion and is focused on providing a functionally identical pure Java application [21]. The software company is driven towards providing Progress 4GL users freedom from the bond to the Progress Software Cooperation and provides two solutions, one is the traditional option where a team of dedicated developers would rewrite the application and the second is the automated conversion. Some important facts that the whitepaper highlights are the complications of Progress 4GL which are, inconsistent 4GL syntax that in many cases is arbitrary or confusing to users; the large amount of implicit behaviour that causes many material aspects of a program to be hidden and also the lack of modern features for structuring and/or reusing code causes heavy reliance upon shared variables, cut and paste, the pre-processor and other techniques which increase maintenance effort.

2.3. 4GL Reverse Engineering

Apart from actual software that are available for conversion and migration of 4GL systems, there are research that has been carried out on reverse engineering 4GL systems so that the design can be extracted out from them. First of such reverse engineering research is the Automated Reverse Engineering of Legacy 4GL Information System Applications using the ITOC Workbench [22]. The paper describes the process of extracting information from the Ingres Relational Database application. This data is then converted into Oracle CASE repository elements. Each of the steps described in the process produces an object in the ITOC analysis schema and are provided as input to the next process. Overview of this process is shown in Figure 2.3.

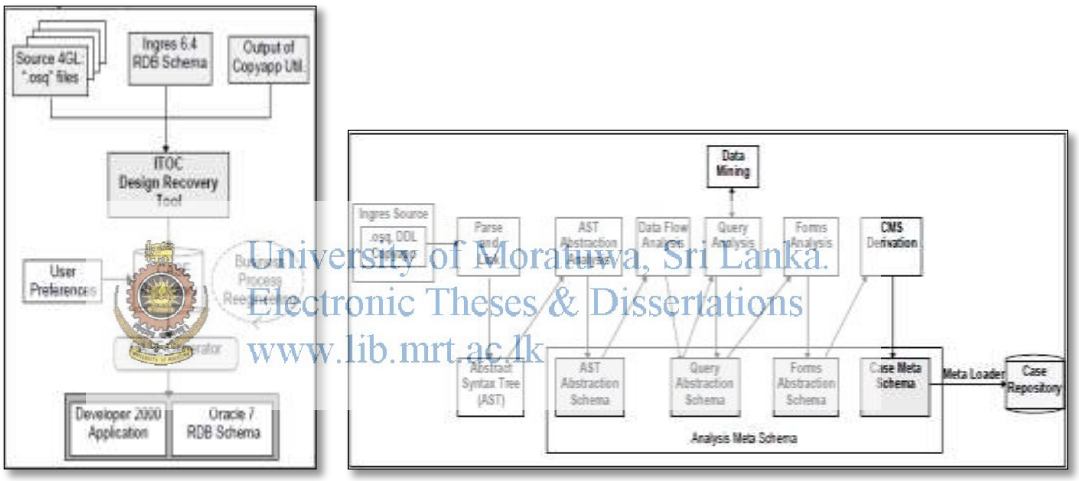


Figure 2.3: ITOC Design Recovery Process: Source [22]

There are more studies done on 4GL legacy systems that are dependent on the vendor and are not automated and require manual migration. These are often not economically feasible and therefore to reduce costs of the migration or re-engineering, it is required to develop tools to automate processes and assist the developers [23], [24] and [25].

The design recovery process of a Logistical Wholesale System [23] is given in the Figures 2.4 and 2.5. This process contains reverse engineering the Magic 4GL application and presenting different architectural views to the developers. The views extracted are used to determine the functional components and logical components to map the entity relationships. This reverse engineering process first extracts the facts

using the Columbus reverse engineering methodology [30] and produces the schema that determines the structure of the Abstract Semantic Graph (ASG). As the second step the design is recovered by using analysis techniques to gather information on the views: Physical View, Call View, Menu View and the User Rights View. The third step is to recover the Database Design. Here the tables and their relationships and the definite CRUD operations are clearly identified.

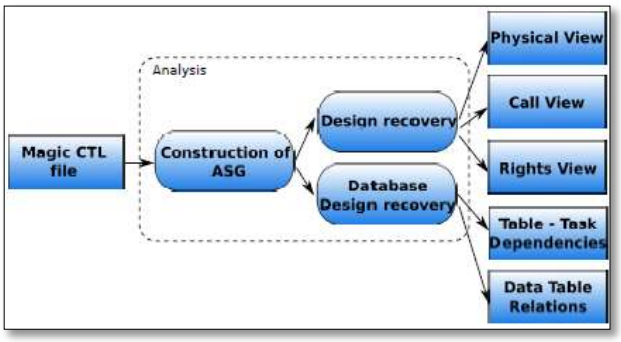


Figure 2.4: Design Recovery Process of a Logistical Wholesale System: Source [23]

Figure 2.5: Design Recovery Tasks: Source [23]

Another common issue with 4GL systems is that they are hard to modify without a specific skill set and knowledge and are expensive to maintain on the long run and are quite difficult to integrate with new technology. Therefore, upgrading or evolving a legacy 4GL system is difficult. To address this issue, the paper A Transformational Approach for Legacy System provides guidelines for an optimal transformation of legacy systems into Unix-RDBMS architectures [24]. This paper stresses that the reuse of the existing system components as much as possible to recover the assets data and the business logic. For the problem at hand re-using components is not acceptable, however, the paper provides a checklist of items to lookout for when transforming such system. Commonly they are to avoid short term solutions, avoid partial solutions, avoid altering the code logic, avoid creating new

bugs in the code, avoid environment emulation solutions, and avoid emulation of data formats and to normalize the data implementation design. Additionally, it insists on paying particular attention to the indexed-files and not to alter data access logic inside the code or create unnecessary data.

The similar concept as to why 4GL systems needs to be migrated are depicted in the paper that describes automation in cross-platform migration shown in Figure 2.6 [25]. In addition to the methodology, the paper describes possible version conflicts that can occur during conversion.

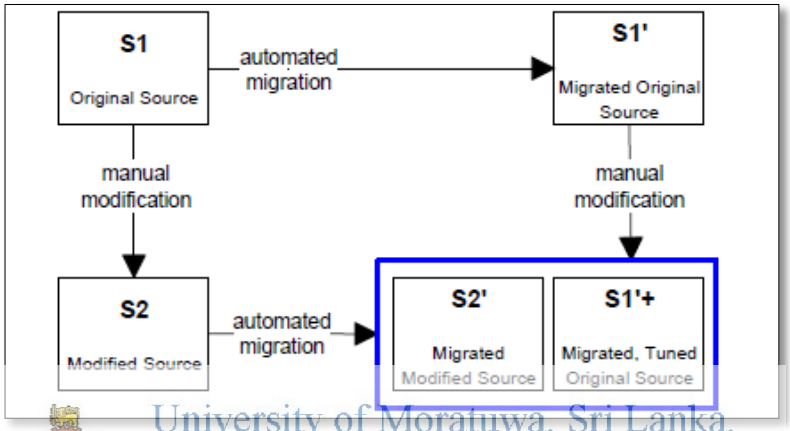


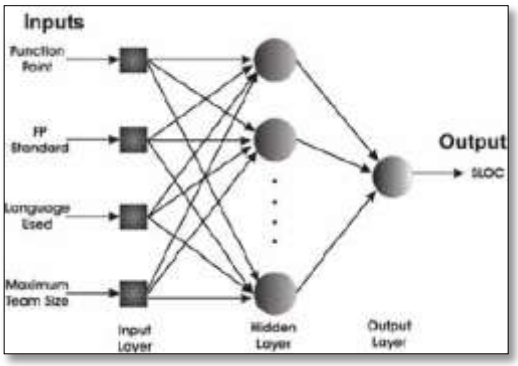
Figure 2.6: Version Conflict in Migration – Source [25]

2.4. 4GL and Machine Learning

Another area of research is on the use of Machine Learning on 4GL code. While there were no research papers to be found on the area of Machine Learning on 4GL to interpret the code, several researches on Code and Effort estimation were available [27], [28], [29] and [30].

The effort estimation paper that uses Statistics and Data Mining aims at two goals: present a comparative analysis about statistic techniques and data mining methods; and to build different approaches to predict the maintenance time of 4GL programs from the metrics considered in this work [27]. Here the authors have focused on predicting the estimates based on the different types of operations used on the code. The research is carried out on historical data collected rather than reading the code to estimate the metrics.

The two neural network approaches to estimate lines of code both require the function points of the 4GL code known before applying the prediction. Therefore does not process the 4GL code itself to get to the results [28] and [30]. The method followed by the authors is shown in Figure 2.7.



4GL System	Actual Development Hours	Estimated Development Hours	Percentage Error	Category
1	5 027	5 015	-0.24%	Training
2	1 680	1 652	-1.68%	Training
3	13 300	13 085	-1.62%	Training
4	98	269	174.41%	Training
5	588	377	-35.89%	Training
6	450	262	-41.69%	Training
7	40	62	55.42%	Training
8	81 270	62 236	-23.42%	Training
9	35 000	34 832	-0.48%	Training
10	240	240	0.11%	Training
11	100	89	-11.08%	Test
12	4 864	4 429	-8.95%	Test
13	1 200	1 184	-1.30%	Test
14	2 500	2 144	-14.24%	Validation
15	1 400	1 396	-0.27%	Validation

Figure 2.7: Neural Network Model: Source [28] and [30]

Comparison of several machine learning methods for Software Effort Estimation (SEE) along with guidelines to choose the correct machine learning model to improve the SEE is explained in the paper "A principled evaluation of ensembles of learning machines for software effort estimation" [29]. This method too however relies on the functional size to derive at the final estimate.

2.5. Systems that Think Rationally

There are many complications and constructs that differs programming languages from Natural Languages. For example, in order to gather the context of a program one may be required to read the entire code following the entire call stack to get the outcome where as in Natural Languages the scope is much restricted to a sentence made up of several combinations of words that can more easily be translated into another language. Therefore, prior to building a system to understand a programming language, the understanding on machines that can think is needed. Following sections provides some well-known examples followed by an overall overview of the literature on such expert systems.

2.5.1. Google Translate

Google Translate is one of the most popular and well know Natural Language processing system that uses Statistical Machine Translation to generate translations based on patterns found in large amount of text [31]. Google Translate uses the same concept as any beginner learning a new language: it first starts with set of vocabulary words and grammatical rules to construct sentences. However, since learning all rules to a language is difficult and since there are exceptions to these rules as well as to having to cater to multiple languages, Google Translate try to discover the rules by itself. In order to generate these rules, it makes use of millions of text and documents already translated by human translators. Books, documents and Webpages from libraries and organizations are scanned by the system to recognize statistically significant patterns. Following this process many number of times Google Translate computer system has become one smart computer [31].

2.5.2. Moses: Statistical Machine Translation System

Similar to Google Translate, Moses is an open-source Statistical Machine Translation system that infers translations between two languages by using occurrences of words and segments (phrases) in stream of parallel input [32]. Initially developed by a student of University of Edinburgh, the system consists of two major components: The Training Pipeline which is a collection of tools that builds the machine translation model from raw data and The Decoder which is a C++ application that translates a source language into a target language when the machine language model generated from the training pipeline is provided. The training model tokenizes text and converts them to standard case to apply heuristics to remove misaligned data and generates word alignments (Figure 2.8). Using these alignments, phrase to phrase alignments or hierarchical rules are extracted and to estimate probability corpus-statistics are applied on these rules. [32] The Decoder's job is to find the highest scoring sentence to get the target language translation.

According to the model generated by the Moses team, the best English output sentence $\mathbf{e}_{\mathbf{best}}$ given a foreign sentence **f** is as follows (Figure 2.9):

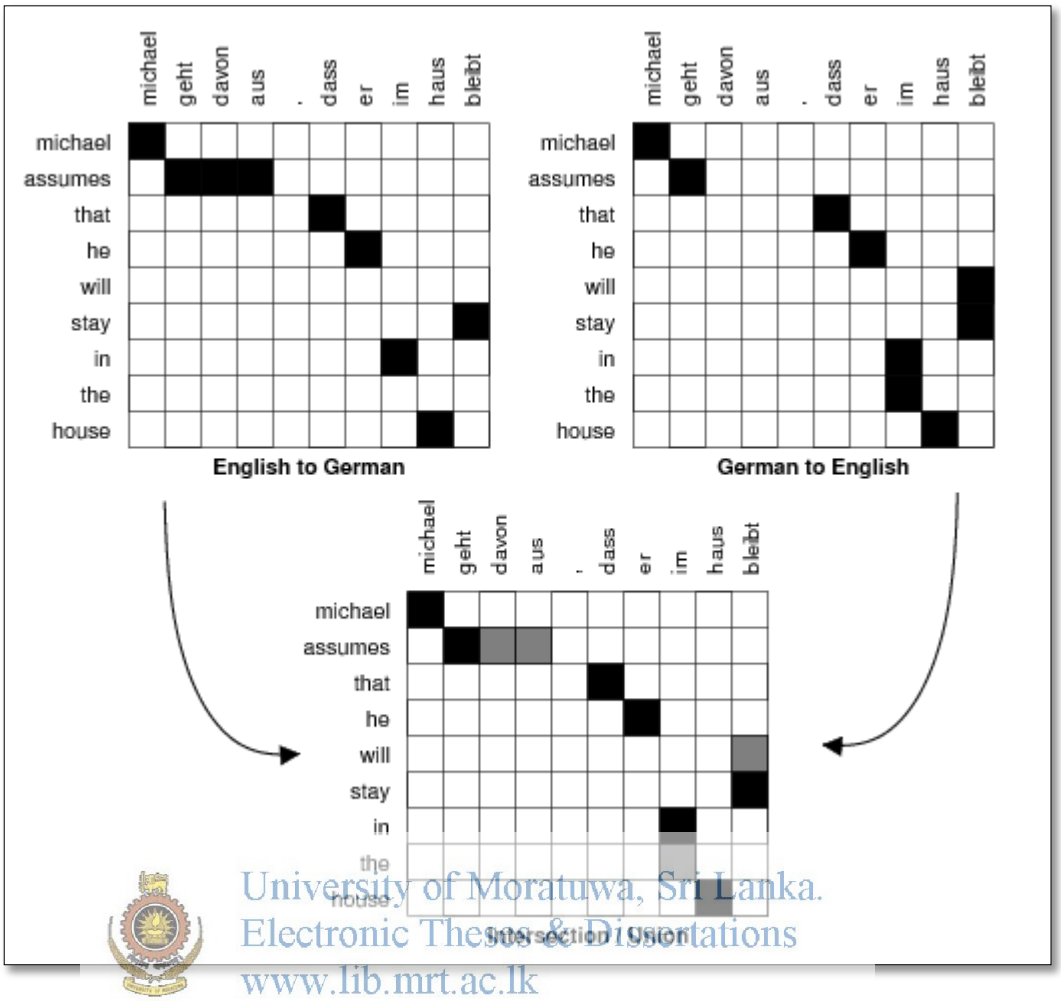


Figure 2.8: Word Alignment. Source [32]

$$e_{best} = \text{argmax\_}e\ p(e|f) = \text{argmax}_e\ p(f|e)\ p\_LM(e)\ \omega^{length(e)}$$

where $p(f|e)$ is decomposed into

$$p(\underline{f}_1^I|\underline{e}_1^I) = \Phi_{i=1}^I\ \varphi(\underline{f}_i|\underline{e}_i)\ d(start_i, end_{i-1})$$

Figure 2.9: Moses Best English Output Sentence Model. Source [32]

2.5.3. Google Prediction API

Google Prediction API is a platform that provides pattern-matching and machine learning capabilities where based on these users can create their own applications to predict user preferences based on past viewing habits, or categorize email as spam or determine the nature of comments to be positive or negative by analysing posted comments or guess daily expenditure based upon spending history [33].

2.5.4. Creating New Language Translation for a Rulebase

A more simple and straightforward translation system can be created by customizing the Oracle Policy Modelling tool where translation document is created along with all rule-base elements such as attributes, screens, messages, events and general metadata which requires to be translated for deployment [34]. The Oracle Policy Modelling supports rule authoring in any language and allows user to define the language parser for writing rules and region based formatting for dates, numbers and currency values.

2.6. Example Based Machine Translation (EBMT)

Example Based Machine Translation (EBMT) is a translation and machine learning mechanism where knowledge is instituted using a database of examples and these examples are used to translate new source text. Examples are pre-translated sentences [35-43]. Overview of the system is shown in Figure 2.10. The interior workflow of an EBMT is shown in Figure 2.11.

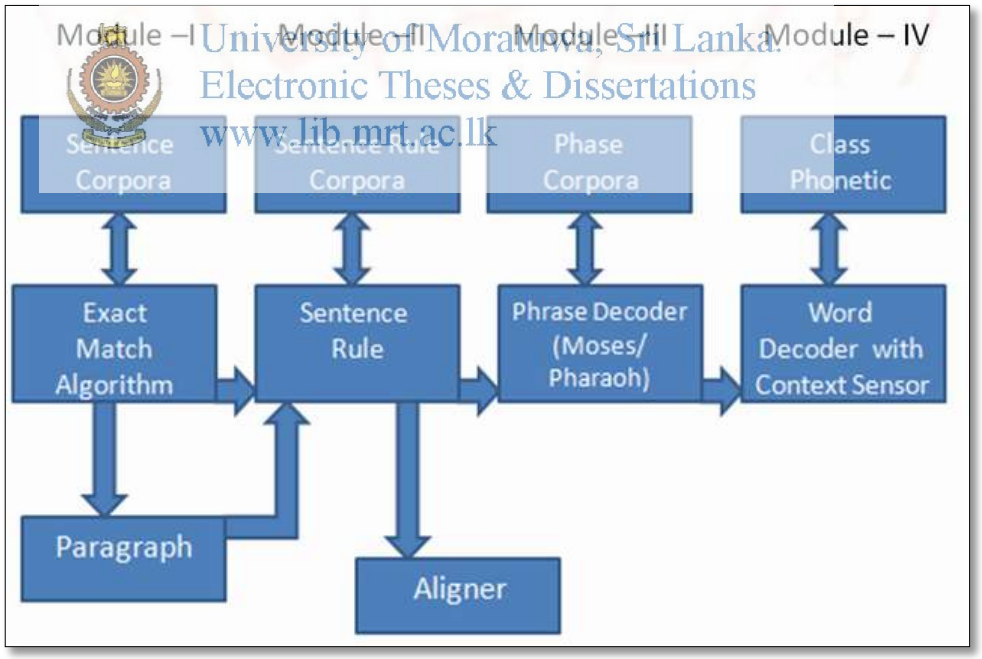


Figure 2.10: Understanding of Example Based Machine Translation (EBMT) system to create translation system. Source [42]

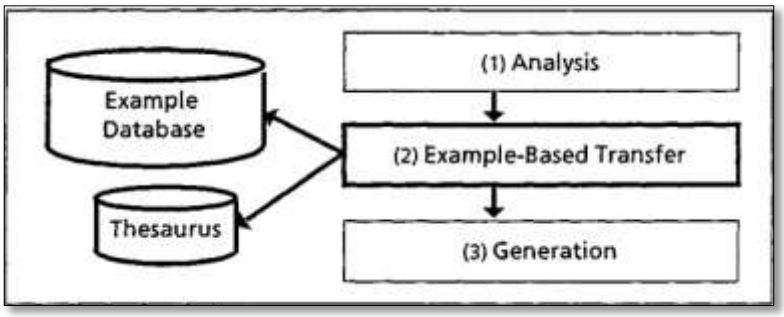


Figure 2.11: EBMT System Configuration. Source [35]

2.7. Rule Based and Expert Systems

One technique of creating AI systems is Agent-based Software Engineering. Many researches has been conducted in this area and a collection of these revised papers are available in the Agent-Oriented Software Engineering book published by the First International Workshop in 2000 [44]. Most of these agent based systems are distributed production systems. Similar methodology should be applicable when creating an application to learn a Programming Language as well. Following figures 2.12, 2.13 and 2.14 depicts the common formation of agent based systems that can be applied here.

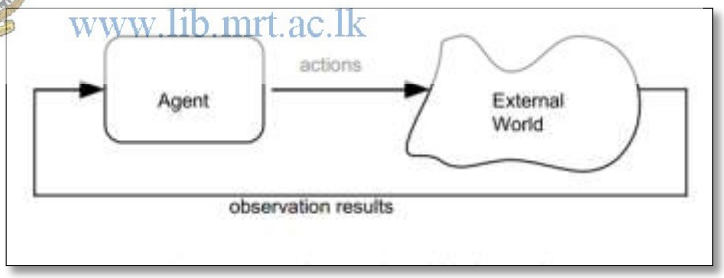


Figure 2.12: Agents Acting on a Dynamic Environment: Source [44]

Figure 2.13: Strategic Decision Making to Arrive at a Solution: Source [44]

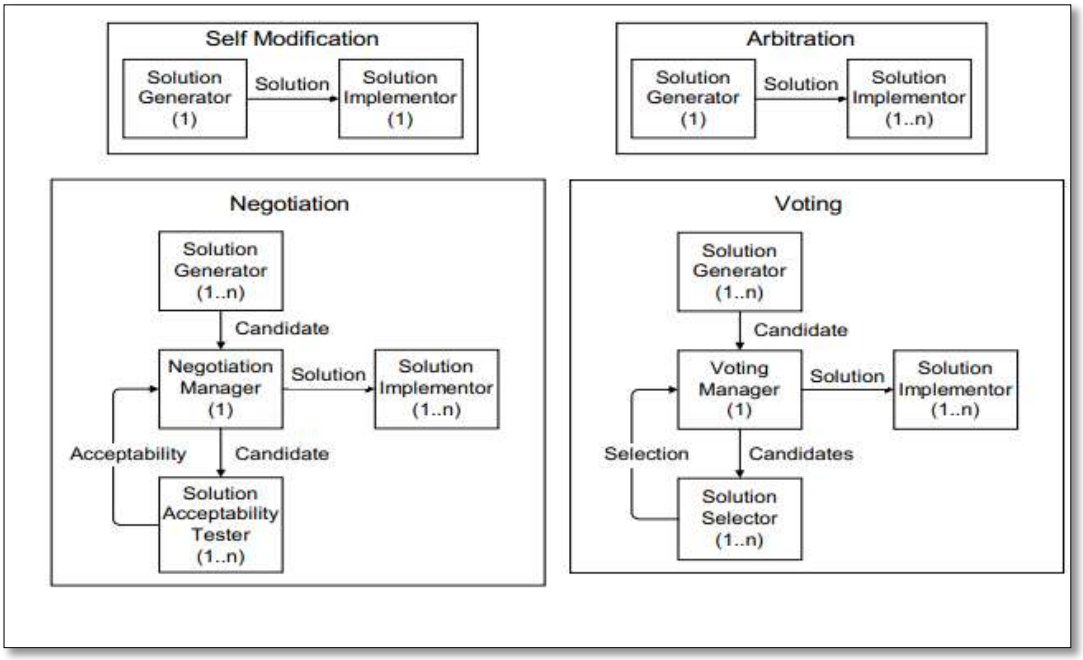


Figure 2.14: Roles of and Agent: Source [44]

Another common approach used for Natural Language processing is through Rule Based Systems [45], [46] and [47]. There are researches that are done on rule-based systems that are capable of adapting to a particular application domain [44]. There are statistical techniques that have been followed in order for a machine to be able to translate one language to another [48]. The researches have constructed systems that are capable of speech recognition through identifying linguistic rules that determines the complexity of a language and constructing a coherent framework where the rules are assembled to recognize the speed. Similarly, for programming languages, it would be important to first identify the "linguistic" of the language per-se before creating the rules based upon them which the system can apply on.

2.8. Intelligent Compilers

ConTraST Transpiler is a configurable Specification and Description Language (SDL) transpiler with its own runtime environment [49]. A Transpiler can be defined as a Transcompiler or Source-to-Source Compiler or in simple terms a translator for programming languages and the SDL, Specification and Description Language is a powerful and comprehensive language that contains four constructs, the *Core* which is the minimum language coverage to provide communicating, the

Static1 the extension with common language features within processes covering, the *Static2* the language extension with services, inheritance and all other transition triggers and finally *Dynamic* that covers those language features which make use of dynamic memory or extension to the state tree during runtime.

The paper introduces a transformation method of an SDL specification into a C++ representation. This approach is similar and the reverse of the current proposed process where instead of a SDL specification as an input, the system is to output a similar construct after analysing and understanding Progress® 4Gl code instead of C++. The authors first perform a syntax analysis of the specification and derive an object oriented syntax tree. Then the syntax tree is transformed into process types and instances and all SDL expressions are transformed to an object oriented format. Once these are gathered the potential for possible optimizations or further transformations is determined. This resulting syntax tree is then used for the code generation. Finally, the system file or the main program and the makefile for compilation are generated.

Figure 2.15 shows the high level architecture of the system that creates code out of the SDL.

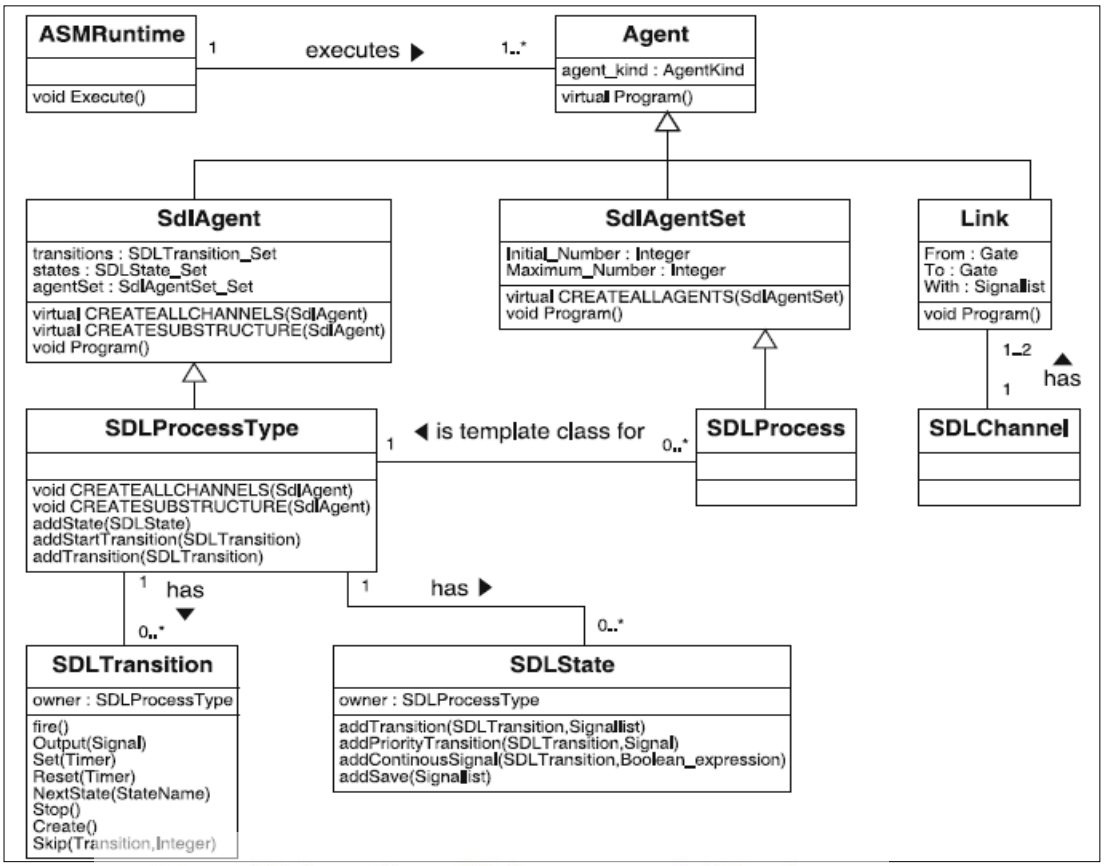


Figure 2.15: Architecture of the ConTraST runtime environment and the abstract syntax: Source [49]

Another category of intelligent compilers is the pattern-matching compiler introduced in 2001. This pattern-matching compiler contains a Yacc-like pre-processor and does not depend on a given term representation and it accepts implementation of terms (or term like data-types) of yet existing applications. This tool is well-suited for industrial use and the implementations of rule-based languages since it permits to define and execute rewrite rules upon those types [50]. In order for the proposed system to read the 4GL code an Intelligent Compiler will be required to tokenize the program code and identify key words, statements and other language constructs in order to extract the summary of the logic presented in the code.

2.9. System Requirements

With more research into similar approaches followed for Natural Language processing, a common methodology can be derived to approach the problem of Extracting 4GL Programming Language Constructs through Machine Learning.

There are also very little research that has been done in Programming Language translation itself, yet alone to create an Expert System to do the translation. The closest Programming language conversion research that are published is between FORTRAN and C [52]. There has also been a research on how rules can be mapped out of a programming language in 1978 [53]. Each of these is restricted to a different type of language and the closest research is the research done on procedural and 4GL language generation and migration tools that have looked into structure of the these languages [54], [55] and [56]. However, translation to 4GL would require finer breakdown of the language constructs and proper construction of the knowledge base and the machine learning techniques. The book, Artificial Intelligence: A Modern Approach, highlights three key areas that an expert system should have and which would be the key points on which the proposed methodology is built up to achieve the goals of the project.

1. **Natural Language Processing** to enable it to communicate successfully in a human language
2. **Knowledge Representation** to store information provided before or during the automated reasoning to use the stored information to solve the problem
3. **Machine Learning** to adapt to new circumstances and to detect and extrapolate patterns

2.10. Summary

With the results of the above mentioned related work areas, it is certain that the problem domain is a frequently looked at issue though it is not for the same 4GL language and out of the many existing solutions available, the proposed system to learn and interpret Progress® 4GL has not been attempted. Further with the references to the Natural Language Processing techniques to Expert Systems to the intelligent compilers a path to achieving the project goal seems satisfactory.

3. APPLICATION OF MACHINE LEARNING FOR EXTRACTING PROGRAMMING LANGUAGE CONSTRUCTS FROM 4GL LEAGACY CODE

The basis of the proposed solution in order to achieve the aforementioned thesis statement to create a smart application to interpret 4GL code was built upon the popular Deep Learning [61] concept for Natural Language Processing where researches are looking at how to create a mind.

3.1. Deep Learning - How to Create a Mind?

For over 2000 years scientists and philosophers have been studying intelligence looking into how the human brain see, learn, remember and reason what has to be done in order to artificially reproduce that intelligence. Solving this puzzle Artificial Intelligence (AI) has become a discipline and in times a popular science [58].

Artificial Intelligence has ever since evolved from the first stored program executed in electronic memory in 1946 to Alan Turing's Turing Test to today's complex AI systems that are capable of making decision through inference which shows signs of bringing Science Fiction pages to reality. Figure 3.1 summarized the timeline of AI break through achievements.

The vast field of Artificial Intelligence has many variations. In order to simply understand what type of AI system is required to build out of this Stuart Russel & Peter Norvig advices on two questions to consider in the book AI: A Modern Approach [58].

1. Are you concerned about Behaviour or Thinking?
2. Do you want to model humans or particular work from an ideal standard?

Based on the possible answers for the above questions the following categorization of AI systems is derived.

1. Systems that can think like humans
2. Systems that can think rationally
3. Systems that can act like humans
4. Systems that can act rationally

With this understanding the proposed approach for the research is derived to be a system that can think rationally as depicted in Figure 3.2.

There are many AI systems built in this category where machines are capable of analysing, attempting to understand and even producing translations in human (Natural) languages. Best known machine translation system is the Google Translate™ translation service. Taking a step further, IBM® Watson™, the reigning JEOPARDY! Champion is capable of answering questions [59], [60]. There are also knowledge extraction systems that are built for auto-coding of medical record data as well as software for online market research. Google News™ service also falls into this category and is capable of classifying news into buckets matching their relevance.

Figure 3.1: AI's Evolution – Source [61]

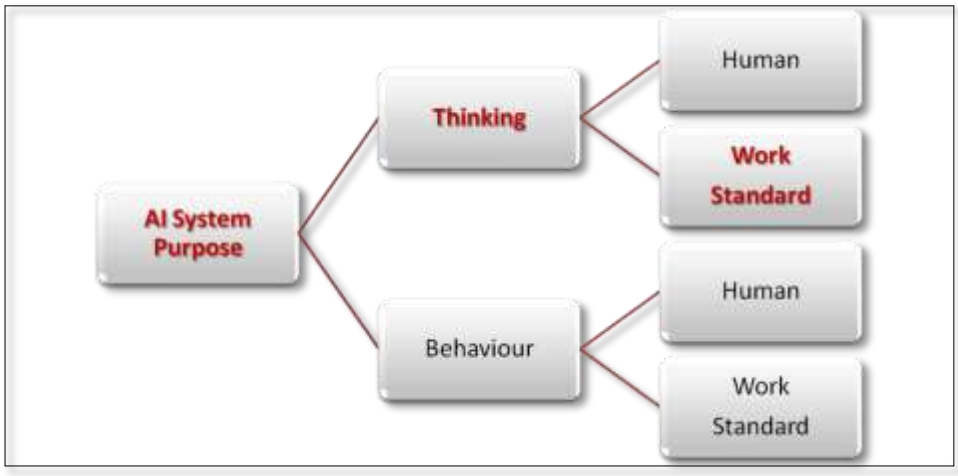


Figure 3.2: AI System Categorization

Deep Learning is a concept which first emerged in 1974 by Harvard scientist Paul Werbos [61] and is practiced Google to achieve smarter AI. In the works of Machine Intelligence futurist Ray Kurzwell in building a truly intelligent computer which can understand language and then make inferences and decisions on its own, he describes training a machine as a child learning a language for the first time. [61]

"Some of today's artificial neural networks can train themselves to recognize complex patterns."

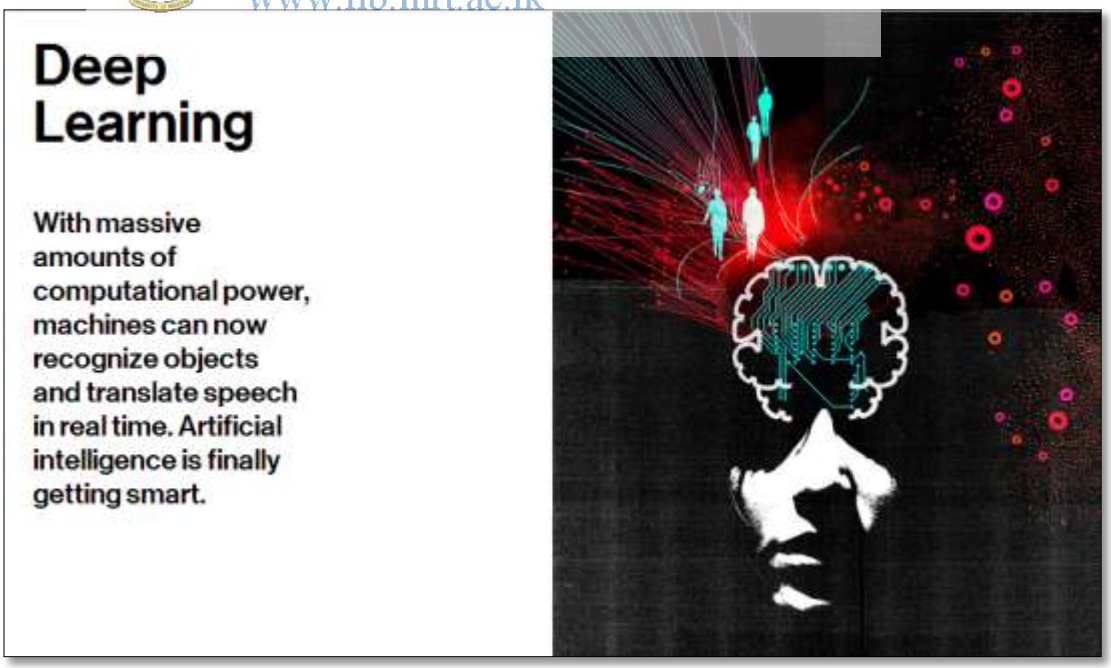


Figure 3.3: Deep Learning - Source [61]

The key point shared by computer scientist and software engineer Jeff Dean of Google Inc. is that even machines should start from raw data and build up higher and higher. Just like children first capture words before building sentences, computers learn to form high level concepts from low level data. They need to be built to make observations and build up a higher level model on its own understanding. [61]

Programming Languages when compared to Natural Languages does not differ much at a high level where both are means of communicate: Programming Languages with machines and Natural Languages with humans. Both contain syntax and semantics to understand the contents. However, Natural Languages existed for thousands of years, and nobody knows who designed the language; but artificial languages are synthesized by logicians and computer scientists to meet some specific design criteria [59], [60]. Out of the various Programming Languages or artificial languages, Fourth Generation Languages or 4GL languages are closer to Natural Languages. Therefore, in order to build a system that can interpret 4GL code could be applied in the same line of concepts of Natural Language Processing.

3.2. Progress 4GL

As described in chapter 2, Progress 4GL is an advanced 4GL language that provides platform for enterprises to develop, deploy, integrate and manage their business applications. As seen in Figure 3.4, Progress Software Company itself has been delivering market-leading software innovations since 1981.

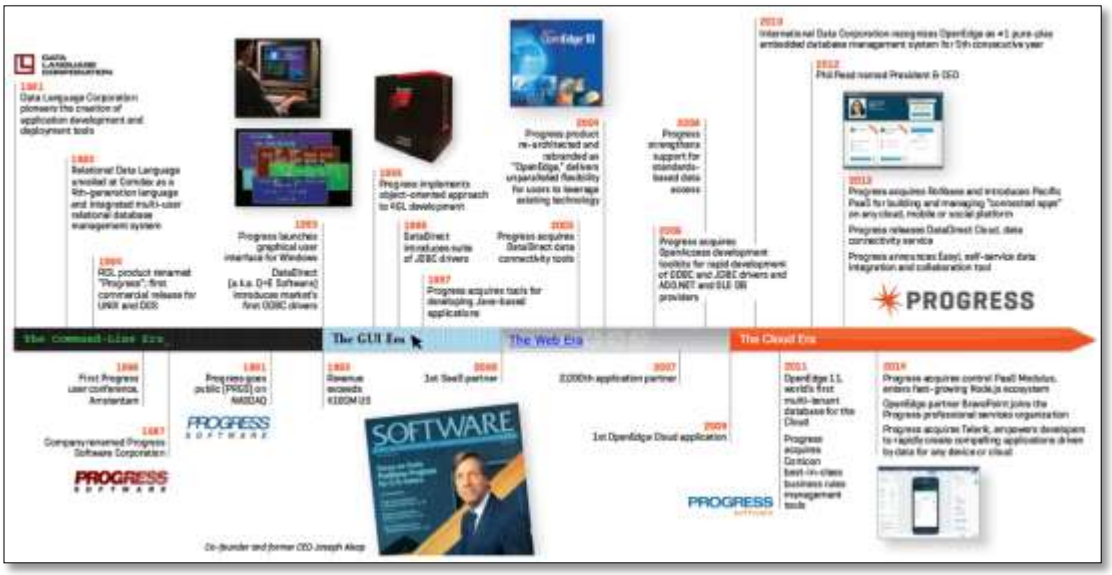


Figure 3.4: Progress Software History – Source [2]

The sample code snippet in Figure 3.5 shows simple data retrieval written in Progress 4GL and in SQL to understand its syntax nature and closeness to the English Language. The proposed system is to take such code as an input, tokenize and analyse the code to extract the logic automatically.

Progress® 4GL

```
FOR EACH customer NO-LOCK
WHERE customer.cs-name BEGINS  "I":
   DISPLAY customer.cs-id customer.cs-name.
END.
```

SQL

```
SELECT customer.cs-id customer.cs-name
FROM customer
WHERE customer.cs-name LIKE 'I%';
```

Figure 3.5: Sample 4GL Code

In order to build up the proposed solution, the 4GL language related research was categorized based on the criteria shown in Figure 3.6 for better understanding and derivation of the rule base for the project. CLIPS AI language is chosen for writing and evaluating the rules.

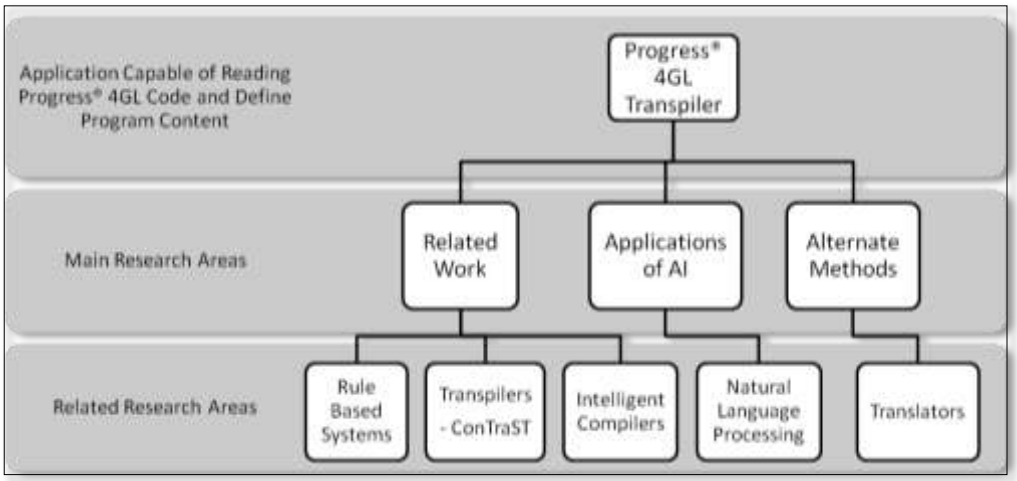


Figure 3.6: Research Categorization

3.3. Java & Machine Learning

Machine Learning is one of the most practical subsets of Artificial Intelligence (AI) that focuses on probabilistic and statistical learning techniques. K-means clustering, fuzzy K-means clustering, K-means, Latent Dirichlet allocation, singular value decomposition, logistic regression, naive Bayes, and random forests are sample algorithms that are useful in learning and predictions [73]. Therefore, in order to incorporate AI to improve the output of the 4GL Programming Language Processing, machine learning tools are required.

There are many open-sourced Machine Learning libraries available that can be integrated with Java based environments especially since Machine Learning is growing as a business tool. Java-ML (Java Machine Learning Library) and JSAT (Java Statistical Analysis Tool) are two such libraries that have a collection of machine learning algorithms implemented [71, 72].

In addition, Apache Mahout on Hadoop platform, Apache Machine Learning Library (MLlib) on Apache Spark platform, MOA (Massive Online Analysis) with Advanced Data Mining and Machine Learning System (ADAMS) and SAMOA (Scalable Advanced Massive Online Analysis) on Apache Storm and Apache S4 projects are available for use for machine learning on Big Data [71]. Among these, Mahout provides a set of powerful mathematical tools designed specifically to use Hadoop to enable scalable processing of large data sets [73, 74]. It provides a fancy e-commerce

API containing algorithms to make predictions, classifications including hidden Markov models which is used to power speech and language recognition and aims at creating a Scalable Machine Learning Library.

Another category of machine learning known as Deep Learning that works with Neural Networks also has Java libraries available for use for project development. Encog is such library that consists of algorithms such as SVM, classical neural networks, genetic programming, and bayesian networks, HMM and genetic algorithms. There is also a commercial grade deep learning library written in Java called Deeplearning4j which is compatible with Hadoop. It provides algorithms including Restricted Boltzmann machines, deep-belief networks and Stacked Denoising Autoencoders [71, 75].

Therefore, Java and Machine Learning integrated together can build smart data-driven applications.

3.4. Tools & Techniques

3.4.1. NetBeans IDE 8.0

For the development of the proof of concept, NetBeans Integrated Development Platform version 8.0 is used in order to develop with Java programming language. Java language was chosen for the following primary reasons:

1. Java is one of the most popular programming languages which is available as a Free and Open Source Software (FOSS) under the GNU General Public Licence
2. There are many libraries available that can help accelerate projects and it supports integration with the following utilities required for the project.
    a. ANTLR (Another Tool for Language Recognition) - parser generator for reading, processing, executing or translating structured text or binary files [62], [63].
    b. MongoDB – Cross-Platform Open Source document based NoSQL database [64].

c. CLIPS (C Language Integrated Production System) - Public domain software tool for building expert systems that has a Clipsjni: Clips Java Native Interface [65].
d. mxGraph - JavaScript Graph Visualization Component using JGraphX [66], [67].
e. Joanju Proparse – Free and open-sourced library parser for Progress OpenEdge ABL [71]
f. Java-ML – Java Machine Learning Library [77]

3.4.2. Proparse

Proparse is a Free and open-sourced parser for Progress OpenEdge Advanced Business Language initially created by John and Joan Green for both .Net and Java [76]. The library consists of classes to parse a Progress 4GL code and generate a parser tree and functions to retrieve and manipulate data in nodes. Use of proparse initially requires configuration of the environment to match the settings of the progress environment that is required to run the code [78]. This project configuration creates the RDBMS schema of the progress environment. Once project schema configuration files are created, instance of the project is created using the RefactorSession class to load the settings. The treeparser class is then called for a file to generate the syntax tree. The library also provides many methods to evaluate the values in the nodes of the syntax tree. These can be used to tokenize and parse into CLIPS to generate the fact and rules.

3.4.3. CLIPSJNI - Clips Java Native Interface

CLIPS is a complete expert system tool which stands for C Language Integrated Production System that provides a development and delivery environment for expert systems. Founded in 1984 at NASA's Johnson Space Centre, it is now widely used throughout due to the following key features provided [52].

1. Knowledge Representation with support for the following three programming paradigms
    a. Rule Based: Represents knowledge as heuristic where specific actions are defined for a given fact

b. Object Oriented: Allows complex systems to be created as modular components
c. Procedural: Similar capabilities as C, Java, ADA and LISP

2. Portability: Since CLIPS is written in C, it can be ported to any systems that has an ANSI compliant C or C++ compiler
3. Integration/Extensibility: Can be embedded within procedural code. Java Native Interface available for JAVA integration.
4. Interactive Development: Standard version available for text oriented development
5. Verification/Validation: Supports modular design and partitioning of a knowledge base, static and dynamic constraint checking of slot values and function arguments, and semantic analysis of rule patterns to determine if inconsistencies could prevent a rule from firing or generate an error.
6. Fully Documented: Reference Manual and Users Guide
7. Low Cost: Public Domain Software

Figure 3.7 illustrates the basic constructs of CLIPS programming. CLIPS programming mainly consists of Facts and Rules. Object Orient paradigm of CLIPS allows users to define fact and rule template objects. Once templates are defined, manipulation commands allow asserting, updating and removing data as well as the constructs.

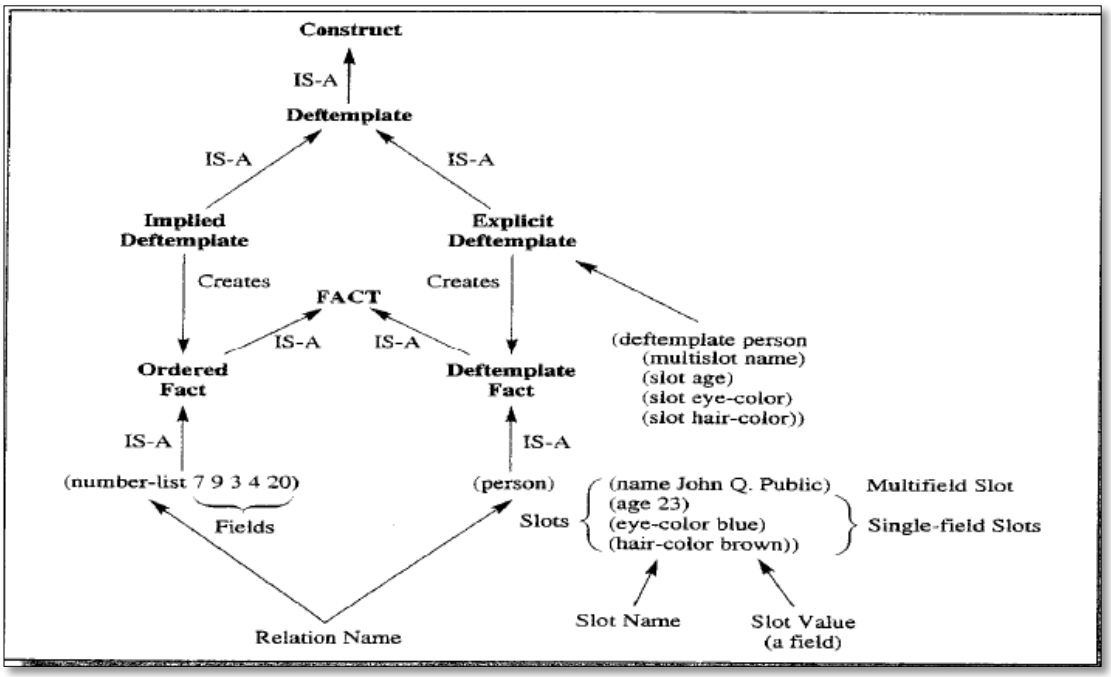


Figure 3.7: CLIPS Construct – Source [68]

Figure 3.8 show a sample CLIPS programming environment with fact and rule definition and evaluation.

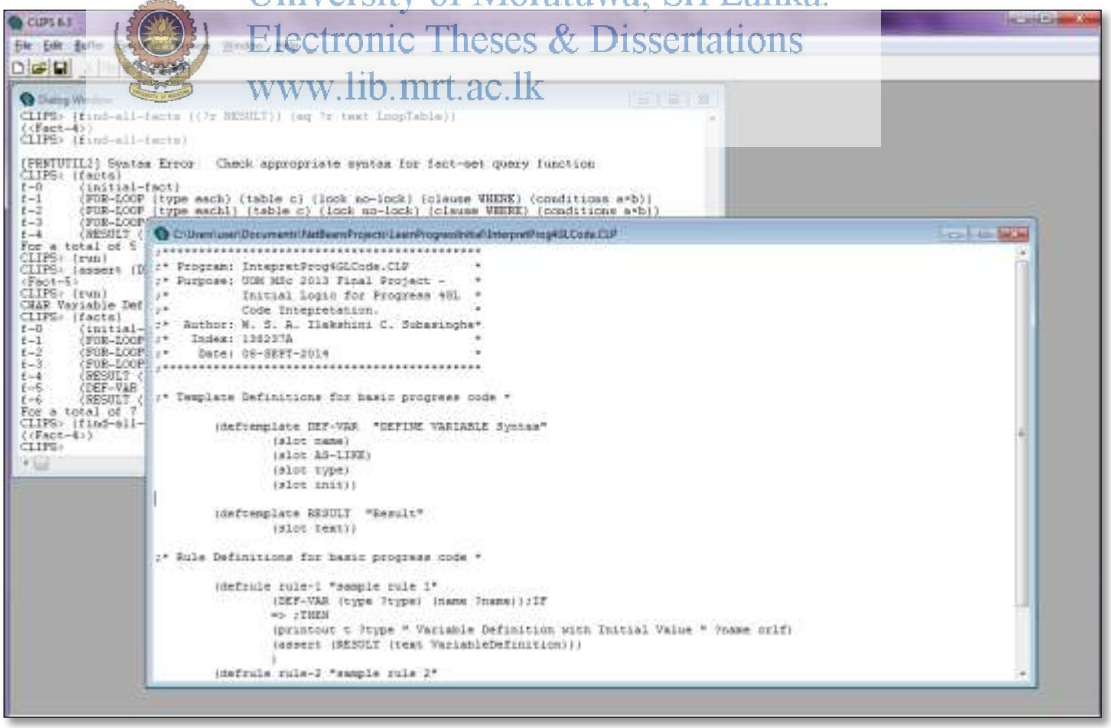


Figure 3.8: CLIPS Dev Platform

3.4.4. Java-ML

Java Machine Learning Library is a cross platform open source library written in Java that consists of a collection of machine learning and data mining algorithms which is directed towards developers to include Machine Learning in their own java projects [77].

Figure 3.9 shows the overview of the different machine learning algorithm classes and interfaces. Integration of these algorithms is simple. Figure 3.10 contains the lines of codes needed to integrate K-means clustering algorithm in a Java program. It uses the FileHandler utility to load the data from the data file indicating which column the class label column is represented. Then an instance of the KMeans clustering algorithm is created with default values (k = 4). The created instance is then used to cluster the dataset where the results are returned as an array of datasets [77].

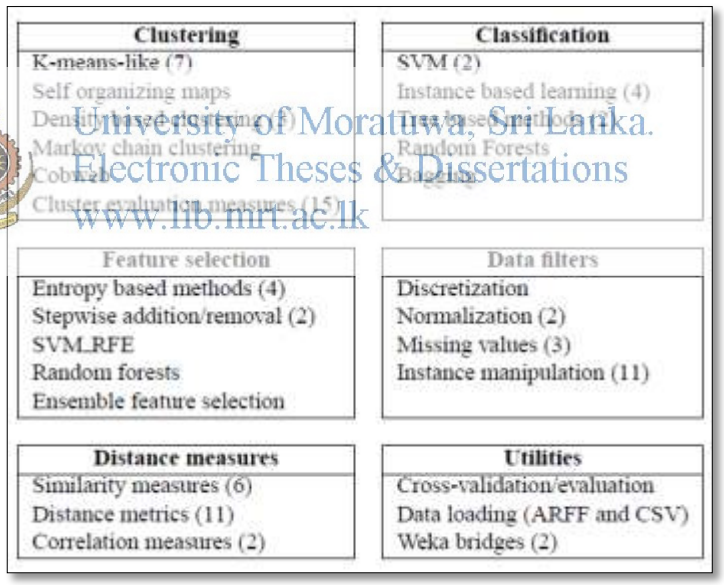


Figure 3.9: Overview of the main algorithms included in Java-ML. The number of algorithms for each category is shown in parentheses. Source [77]

```
Dataset data = FileHandler.loadDataset(new File("iris.data"), 4, ",");
Clusterer km = new KMeans();
Dataset[]clusters=km.cluster(data);
```

Figure 3.10: Java-ML: KMeans integration to Java Code. Source [77]

The Figure 3.11 depicts the lines of codes that is required to cross validate dataset and a classifier using K-Nearest-Neighbor. Similar to KMeans, dataset is first loaded and instance of KNearestNeighbors is created. In addition CrossValidation instance is created passing the KNN instance. Then the data is run though the cross validation which will return map that maps class label to its performance measure.

```
Dataset data = FileHandler.loadDataset(new File("iris.data"), 4, ",");
Classifier knn = new KNearestNeighbors(5);
CrossValidation cv = new CrossValidation(knn);
Map<Object, PerformanceMeasure> p = cv.crossValidation(data);
```

Figure 3.11: Java-ML: Cross-validation experiment for specific dataset and classifier. Source [77]

Once 4GL syntax tree is created and evaluated, Java-ML can be used help in generating a more accurate output flow diagram.

3.4.5. MongoDB

For better processing of the input code program and easy evaluation and interpretation of the facts NoSQL database is used. MongoDB is chosen as it is stated to be the only database that ties together the innovations of NoSQL such as flexibility, scalability, performance while building on the foundation of relational databases such as expressive query language, secondary indexes and strong consistency [64]. The key features of MongoDB are as follows:

1. Flexible Data Model: Document data model makes it easy to store data of any structure and dynamically modify the schema.
2. Highly Scalable: Scale up or scale out horizontally, from a single server to thousands of nodes. Supports deployment in the cloud and across multiple data centres.
3. High Performance: Run high-performance systems at scale. Achieve millions of ops per second for read-heavy, write-heavy, and mixed read-write workloads
4. Expressive Query Language: Provides varied field-level operators, data types and in-place updates. Drivers for just about any programming language make it intuitive to use.

5. Secondary Indexes: Fast, fine-grained access to data, including fully consistent indexes on any field, as well as geospatial, text search and TTL indexes.

3.4.6. ANTLR

ANTLR stands for Another Tool for Language Recognition and provides two main features. [62]

1. Tool that translates user defined grammar to a parser/lexer
2. Runtime needed for the generated parsers and lexers

Hello.g4

```
// Define a grammar called Hello
grammar Hello;
r  : 'hello' ID ;         // match keyword hello followed by an identifier
ID : [a-z]+ ;             // match lower-case identifiers
WS : [ \t\r\n]+ -> skip ; // skip spaces, tabs, newlines
```

Figure 3.12: ANTLR Example – Source [63]


```
$ cd /tmp
$ antlr4 Hello.g4
$ javac Hello*.java
```

```
$ grun Hello r -tree
hello parrt
^D
(r hello parrt)
```

Figure 3.13: Run ANTLR

```
$ grun Hello r -gui
hello parrt
^D
```

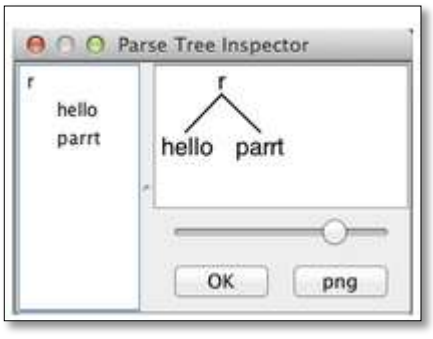


Figure 3.14: ANTLR GUI – Source [63]

3.4.7. mxGraph – JGraphX

In order to output the results of the program evaluation in a graphical flow chart format mxGraph is used. mxGraph contains a Java Swing library version that can be imported to a package called the JGraphX (JGraph 6). [66], [67]

```
// Adds cells to the model in a single step
graph.getModel().beginUpdate();
try
{
   Object v1 = graph.addVertex(parent, null, "Hello,", 20, 20, 80, 30);
   Object v2 = graph.addVertex(parent, null, "World!", 200, 150, 80, 30);
   Object e1 = graph.addEdge(parent, null, "", v1, v2);
}
finally
{
   // Updates the display
   graph.getModel().endUpdate();
}
```

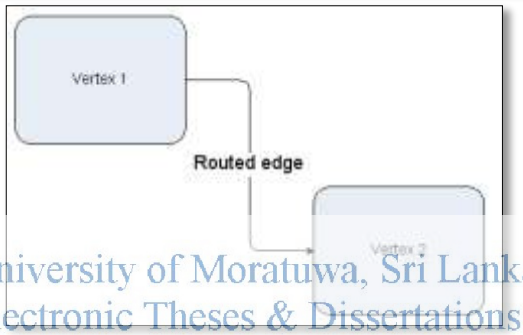


Figure 3.15: mxGraph Example - Source [67]

4. SYSTEM IMPLEMENTATION AND EVALUATION

This section provides the detail description of the implementation of the proposed 4GL interpreter that is capable of processing 4GL code to output the workflow of the program logic in the format of a flow chart.

4.1. System Overview

Figure 4.1 depicts the high level workflow of the system to interpret 4GL code and Figure 4.2 depicts the same with the tools and technologies used. For the reasons stated in Section 3.4 Tools & Techniques Java is chosen as the Language to implement the 4GL code interpreter and is developed using NetBeans IDE.

Application developed provides and interface for the user to input the required Progress 4GL program file. The entered file is assumed to have progress code with no syntax errors in order to build up the initial rule base.

The application then saves each code statement with its line number into the MongoDB instance created. The code is stored so that the functionality can be modularized and separated for better performance.

In order to create a more accurate syntax tree, the Joanju Proparse library for Java is used. The configuration of the progress development environment where the sample 4GL code is created for including the PROPATH, database connections, database aliases etc. are first fully set up. The provided profactor configuration program is then run to generate the schema and other configuration files that are imported to the Java project. Then the proparse tool is used to generate the syntax tree.

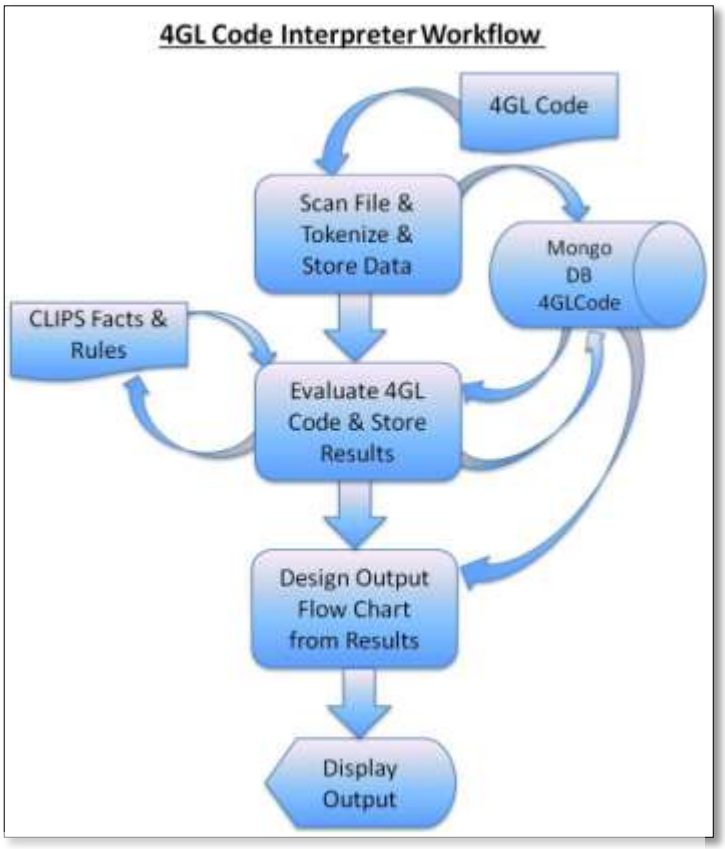


Figure 4.1: 4GL Code Interpreter Workflow

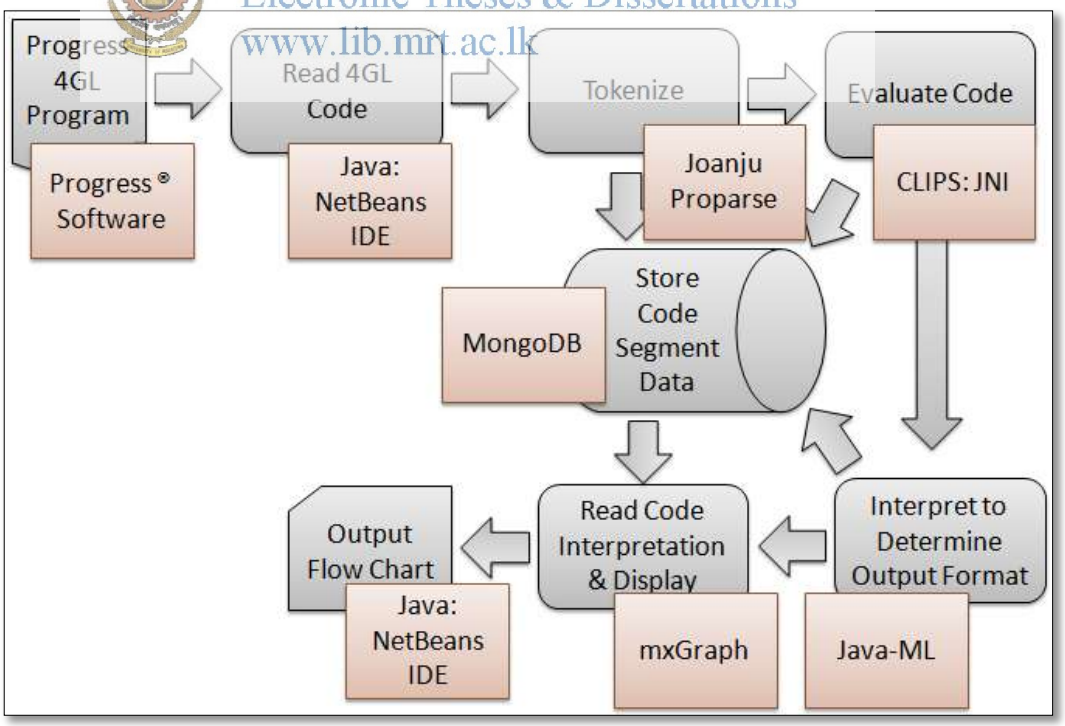


Figure 4.2: 4GL Code Interpreter with Tools & Technologies Used

Once the entire program is stored and the syntax tree is generated by the proparse, the evaluation module starts up the CLIPS environment and runs each statement word to word to compare with its fact base and creates the necessary results matching any rules defined. These evaluated results for each statement or code block is written back to the MongoDB instance.

After evaluating the program and saving the results, the data is queried in order of logic to create dynamic vertex objects using JGraphX for each process. Once entire logic is written to nodes, the edges between the objects are inserted to complete the flow diagram. When completed, the final flow chart of the program logic is displayed using a JFrame.

The results of the evaluated program code to diagram are stored as a data file with appropriate class labels. With this categorization, unknown logic can be predicted with the use of the Java-ML library class instances. These class labels could aid the output flow chart generation to provide a more accurate output without having to manually label each token type in the syntax tree the application comes across.

This basic workflow can then be enhanced so that the CLIPS inference engine can generate and save the rules for future use. To define the language rules and identify which rules to create ANTLR can be used. ANTLR provides similar functionality to Lex & Yacc used with C language.

4.2. Proparse Configuration & Syntax Tree Generation

In order for the Proparse library to compile and build the syntax tree, it is required that the input Progress 4GL code to be compiled and be free of any syntax errors. This includes database schema and propath settings to locate programs called within the progress code as well. Therefore, these configurations need to be generated and imported to the Java project in order for Proparse to run without exceptions. Following describes the steps to setup the configurations.

To generate the schema data and other settings, the Proparse project provides a progress 4GL program that is required to be run in the Progress environment in which the code programs used for the parser are supposed to be run. Therefore, the first step is to start the progress development session with the following setups [78]:

- PROPATH
- Progress Configuration Settings
- Database Connections
- Database Aliases etc.

Once the environment is setup, the provided configuration.p program should be run as follows:

```
RUN prorefactor/configure.p.
```

The program will prompt for the project path settings and the project name in order to generate and save the configuration settings as shown in Figure 4.4.

When the configuration is completed, the program would have created the following files in the provided project location under new folder called "profactor/project/<ProjectName >".

- Progress Properties File


```
batch_mode=false
opsys=WIN32
proversion=11.5
window_system=MS-WINXP
propath=C\:\\Users\\user\\MSC_UoM\\FinalProject\\samples\\prorefactor_config_003\\,.,C\:\\Progress\\OpenEdge\\gui,…,C\:\\Progress\\OpenEdge\\bin
database_aliases=
```

- Proparse Properties File

```
schema_file=C\:\\Users\\user\\MSC_UoM\\FinalProject\\Implementation\\4GLInterpreter\\4GL/prorefactor/projects/4GLInterpreter/prorefactor.schema
```

- Profactor Schema (Entre Sports2000 Database schema of Progress OpenEdge Developer Studio)

```
:: MySportsDB 1
: Customer 11180
CustNum 11183 INTEGER 0
Name 11184 CHARACTER 0
Address 11185 CHARACTER 0
Address2 11200 CHARACTER 0
City 11201 CHARACTER 0
State 11202 CHARACTER 0
Country 11203 CHARACTER 0
:
:
```

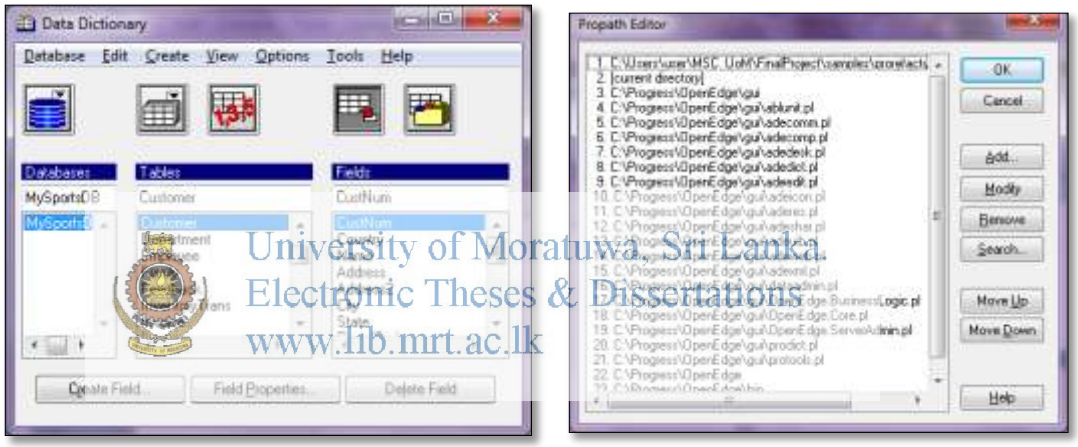


Figure 4.3: (a). Sample Data Dictionary (b.) PROPATH Settings

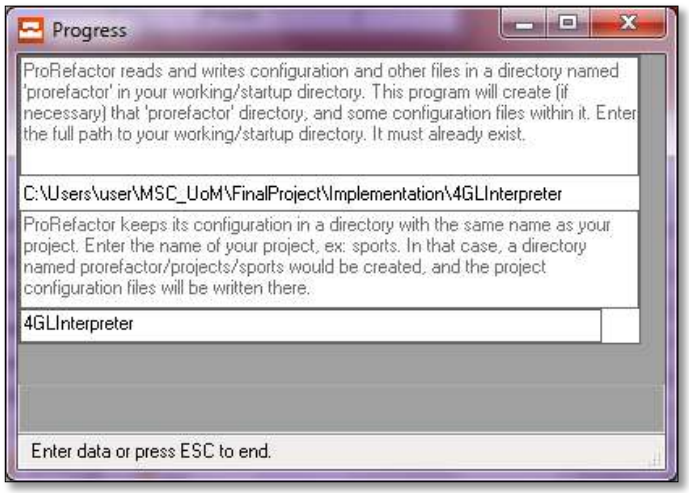


Figure 4.4: Progress Configurations for Proparse

Once the configuration settings files are generated under the Java project source path, the settings are loaded by creating an instance of the RefactorSession class as shown in Figure 4.5 and then the parser can be called to build the syntax tree as shown in Figure 4.6.

```
public InputFileJFrame() {
    /* Create and load the CLIPS environment */
    clips = new Environment();
    clips.load("CLIPS/InterpretProg4GLCode.CLP");
    try {
        RefactorSession prsession = RefactorSession.getInstance();
        prsession.loadProject("4GLInterpreter");
    } catch (Exception ex) {
        Logger.getLogger(InputFileJFrame.class.getName()).log(Level.SEVERE, null, ex);
    }
    initComponents();
}
```

Figure 4.5: Load Configuration Settings with Refactor Session

```
/* User ProParse to generate token tree */
try {
    ParseUnit pu = new ParseUnit(f);
    pu.treeParser01();
    System.out.println(pu.getTopNode());
} catch (RefactorException ex) {
    Logger.getLogger(InputFileJFrame.class.getName()).log(Level.SEVERE, null, ex);
}
```


Figure 4.6: Generate Parser Tree for Progress 4GL Code

4.3. CLIPS Integration & Evaluation

The core of the proposed research lays in the fact and rule generation in order to inference the 4GL code. For the reasons listed in Section 3.4 Tools & Techniques, CLIPS was chosen as the tool to create the expert system for 4GL language processing.

To develop a complete expert system, application should container the following three components.

1. **Fact list** which creates the low level bread crumbs that the system can use to develop into high level concepts.
2. **Knowledge base** which CLIPS provides with three programming paradigms: Rule Based, Object Oriented and Procedural that allows the system to construct actions for recognized patterns.

3. **Inference Engine** which is capable of creating and updating the fact list and validating the facts against the knowledge base to either create/update more facts and rules or perform any specific actions when the rules are matched.

To create the fact and rules CLIPS defines eight types of primitive data types, non-printable ASCII characters, symbols and delimiters.

A fact in CLIPS is capable of saving a chunk of information using its OO concepts. CLIPS program has the capability to create definitions of different types of data formats that are expected. The deftemplate construct provides this functionality to identify the fact class with a name and also provide slots to save different values or data each fact can contain as seen in Figure 4.7.

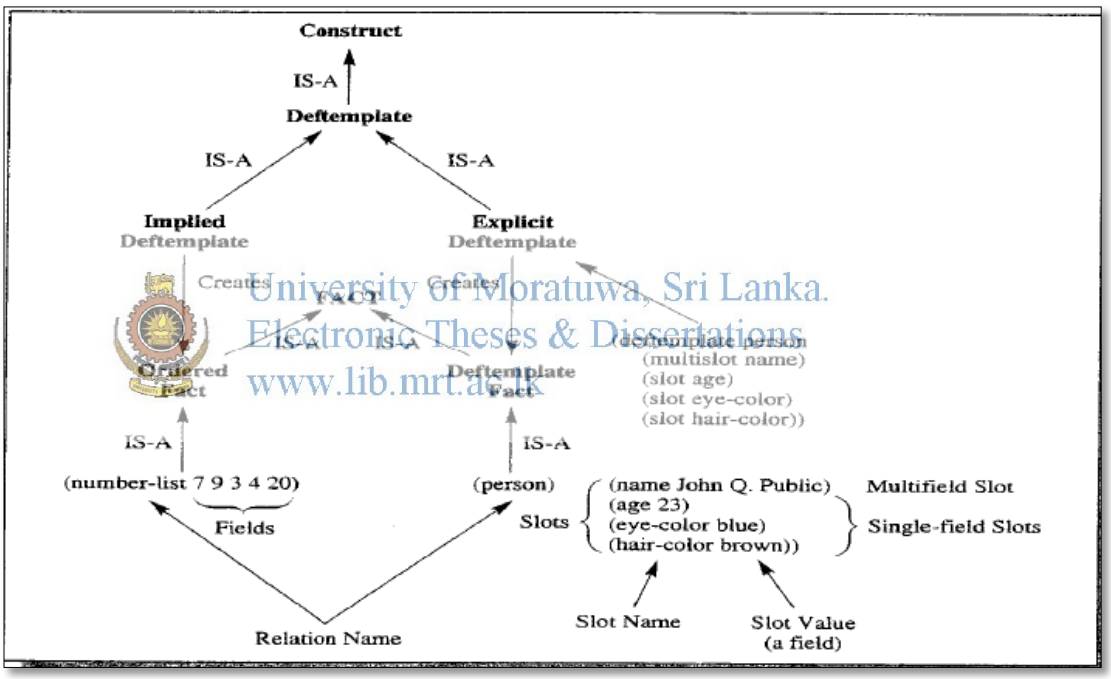


Figure 4.7: CLIPS Constructs. Source [68]

With the use of this construct, first the fact templates are coded into a CLIPS (.CLP) program. For example, the basic variable definition statement of progress 4GL as shown in Figure 4.8 can be converted to a deftemplate format as shown in Figure 4.9.

```
DEFINE VARIABLE variable-name AS primitive-type-name NO-UNDO.
```

Figure 4.8: Progress 4GL Variable Definition

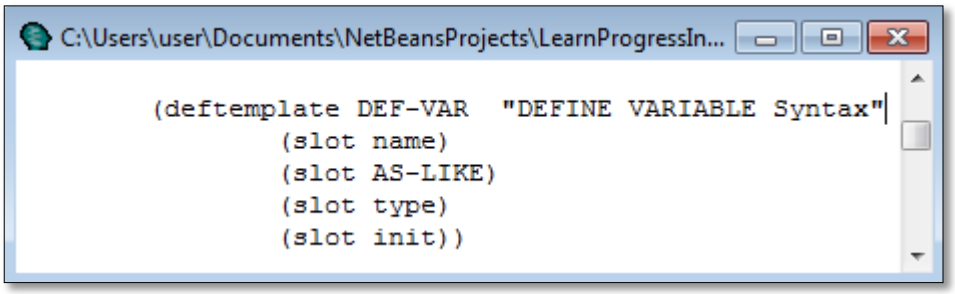
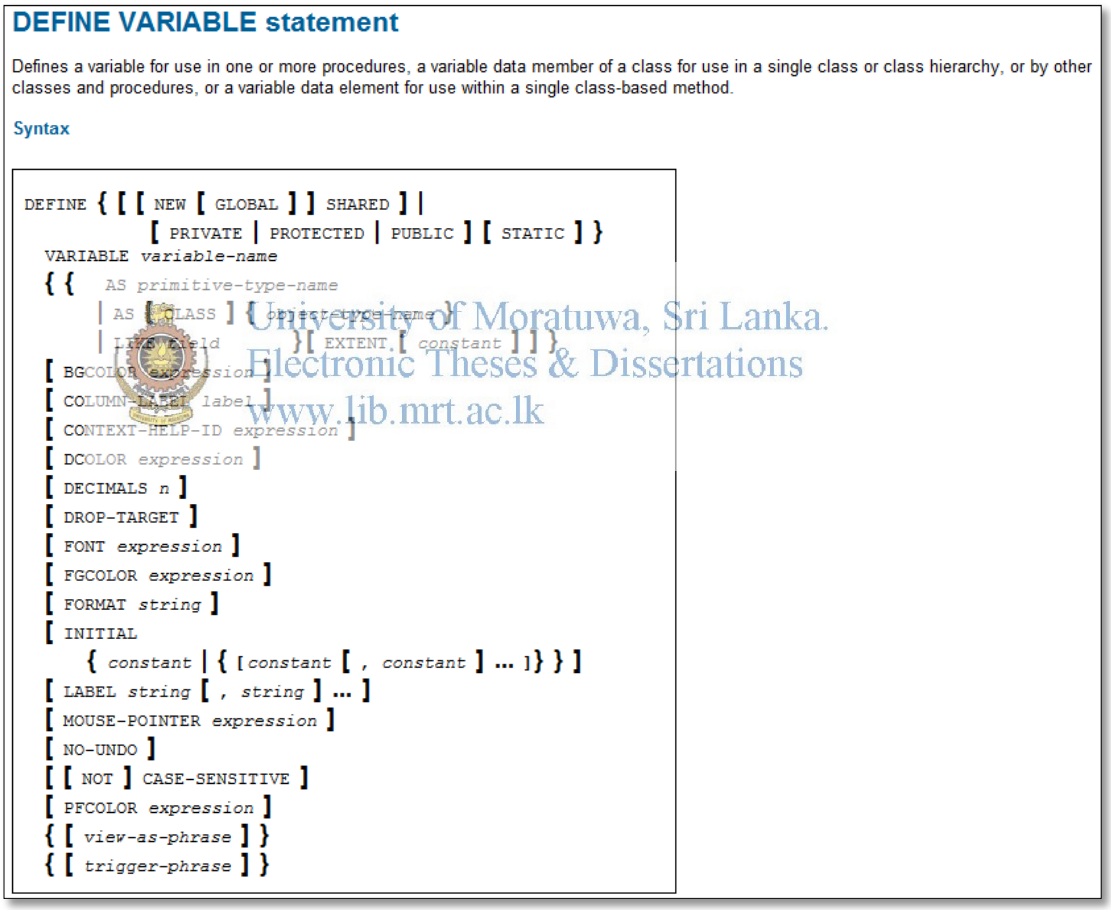


Figure 4.9: CLIPS Template Definition for 4GL Variable Definition

The entire variable definition statement contains many complex constructs. Progress 4GL variable definition syntax is shown in Figure 4.10 provides a glimpse into the complexity of the language and the implementation of all logic into a template.



DEFINE VARIABLE statement

Defines a variable for use in one or more procedures, a variable data member of a class for use in a single class or class hierarchy, or by other classes and procedures, or a variable data element for use within a single class-based method.

Syntax

```
DEFINE { [ [ NEW [ GLOBAL ] ] SHARED ] |
         [ PRIVATE | PROTECTED | PUBLIC ] [ STATIC ] }
  VARIABLE variable-name
  { {   AS primitive-type-name
      | AS [ CLASS ] class-type-name }
      | LIKE field                    } [ EXTENT [ constant ] ] }
  [ BGCOLOR expression ]
  [ COLUMN-LABEL label ]
  [ CONTEXT-HELP-ID expression ]
  [ DCOLOR expression ]
  [ DECIMALS n ]
  [ DROP-TARGET ]
  [ FONT expression ]
  [ FGCOLOR expression ]
  [ FORMAT string ]
  [ INITIAL
     { constant | { [constant [ , constant ] ... ]} } ]
  [ LABEL string [ , string ] ... ]
  [ MOUSE-POINTER expression ]
  [ NO-UNDO ]
  [ [ NOT ] CASE-SENSITIVE ]
  [ PFCOLOR expression ]
  { [ view-as-phrase ] }
  { [ trigger-phrase ] }
```


Figure 4.10: Progress 4GL Variable Definition Complete Syntax - Source [69]

Since the initial focus of the research is to develop a mechanism with which the proposed results can be achieved, the templates are defined for the following selected sample code syntax as shown in Figure 4.8.

- Simple Variable Definition
- Simple Input/Output Parameter Definition
- Display Statement
- Simple For loop which access the progress relational database to retrieve data.

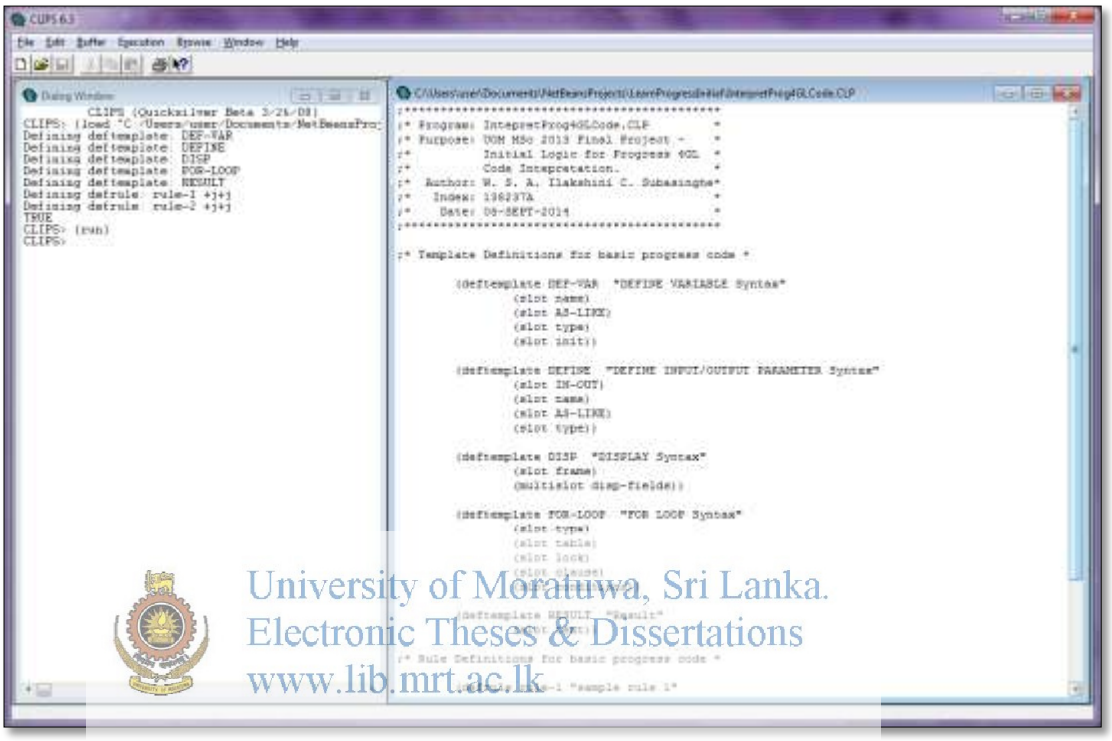


Figure 4.11: CLIPS Template Definitions

Once the templates definitions of the facts are completed, the rules to define what actions needs to be taken when the facts are discovered needs to be coded.

It is required to identify programming logic behind each fact created and how it should be displayed in the output flow chart. They need to be identified as a either a Process (Definition/Data Retrieval) or Conditional Statement or a Display statement and if possible the unique values associated with each statement.

For example of the rule definition, consider the same variable definition statement in Figure 4.8. When detected it is required to display flow chart node with a value similar to “primitive-type-name variable-name defined” and for logging purposes

write to standard output interface the processed rule. Figure 4.11 contains the CLIPS logic for this rule definition.

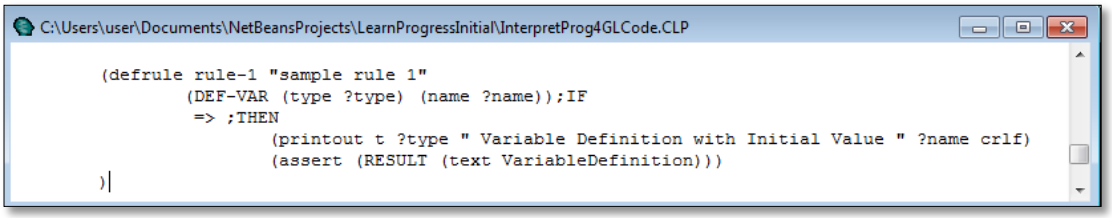


Figure 4.12: CLIPS Rule Definition for Progress 4GL Variable Definition

The rule states that when DEF-VAR template is found, save the slots type, name and other data in variables denoted with the "?" in proceeding the variable name and perform the actions stated in the "=> ;THEN" section.

Any number of commands and statements can be coded in the then section. The example in Figure 4.9 prints out message "Primitive-type-name Variable Definition with Variable Name" and also asserts a RESULTS fact (defined as a template prior) to the CLIPS engine to denote the found record.

Figure 4.13 contains sample rules created to evaluate the sample template facts defined.

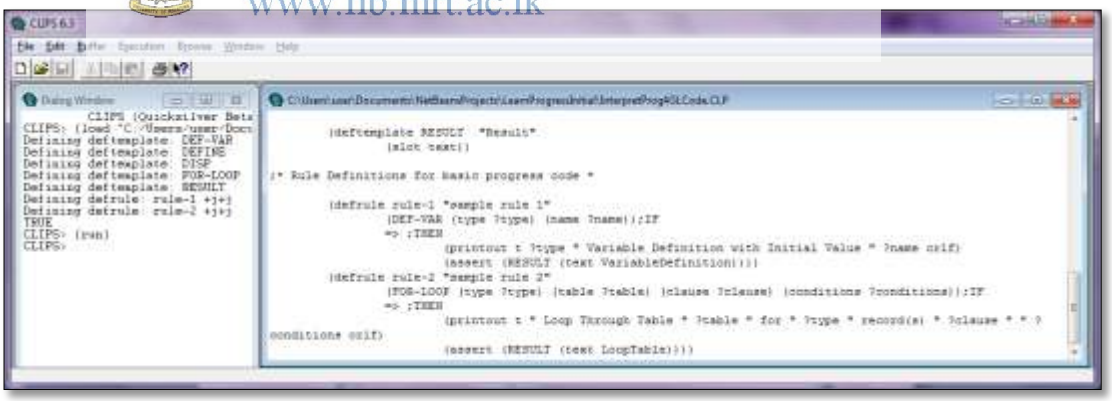


Figure 4.13: CLIPS Rule Definitions

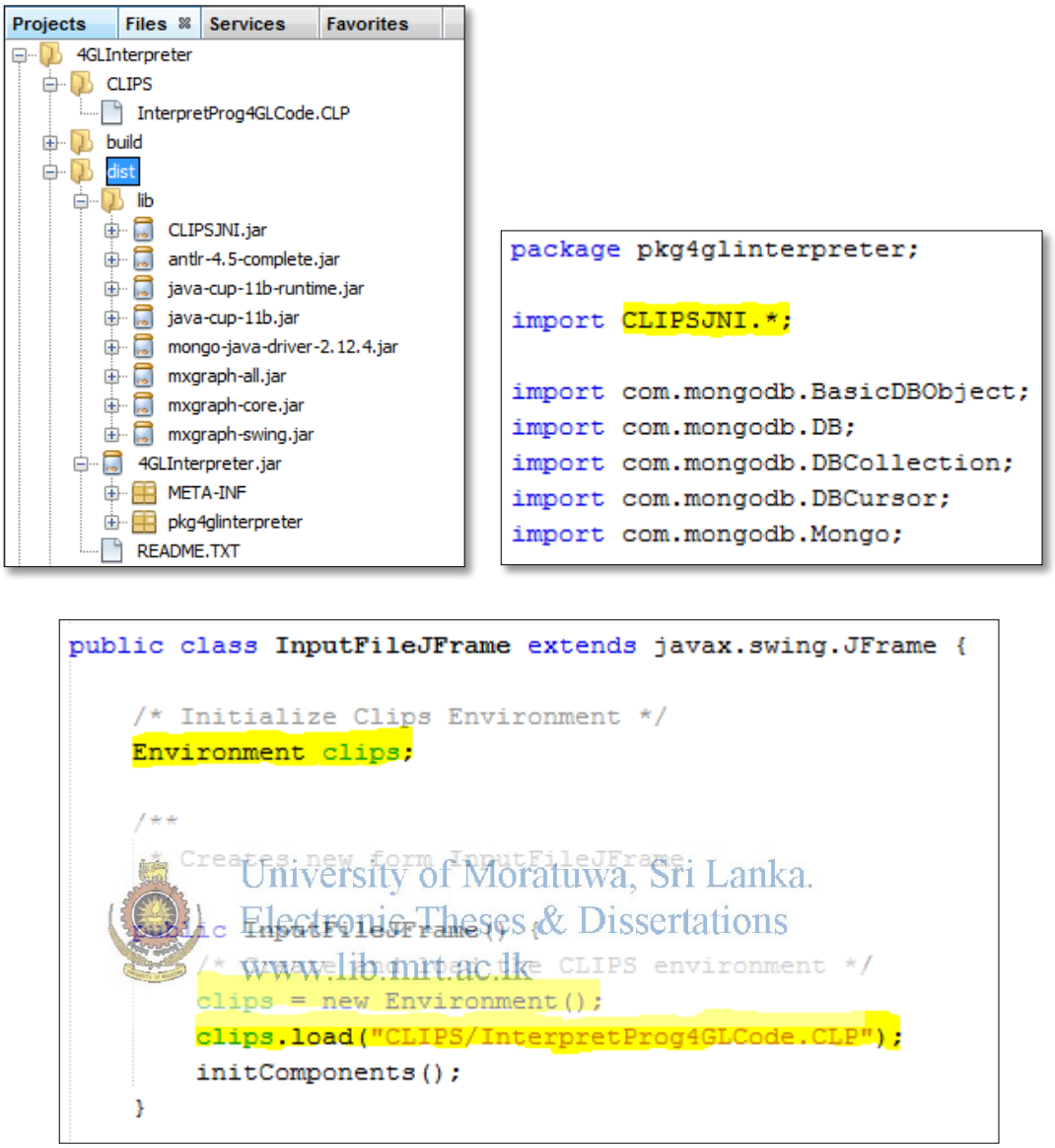


Figure 4.14: CLIPSJNI Integration

Once the CLIPS program is completed it is added to the NetBeans project along with the CLIPS Java Native Interface library in order to use the environment. The CLIPSJNI library is then imported to class where the code is processed.

On load of the frame create the CLIPS environment and load the template and rule definitions from the created .CLP program as shown in Figure 4.10.

```
// begin CLIPS evaluation
/* Tokenize current line for evaluation */
pv = null;
StringTokenizer stCodeTokens = new StringTokenizer(sCurr);
if (sCurr.isEmpty()) {
    sResult = "";
}
else if (sCurr.startsWith("DEF")) {
    try {
        clips.eval("(assert (DEF-VAR (name " + stCodeTokens.nextToken().trim() +
                                    ") (type " + stCodeTokens.nextToken().trim() +
                                    ") ))");
        clips.run();
        pv = clips.eval("(find-all-facts ((?r RESULT)) (eq ?r:text VariableDefinition))");
        if (pv.size() == 0) {
            sResult = "";
        }
        sResult = pv.get(0).getFactSlot("text").toString();
    }
     catch (Exception ex) {
        Logger.getLogger(InputFileJFrame.class.getName()).log(Level.SEVERE, null, ex);
     }
}
else if (sCurr.startsWith("FOR")) {
```

Figure 4.15: CLIPS Evaluation

When evaluating the code text, match patterns to insert facts to the CLIPS environment. The rules defined will process on the clips.run command and create the RESULTS fact for the item. This result is then captured and stored along with the code data in the MongoDB.

When syntax is not matched with the defined template, the program will record the status as a Syntax Error.

This evaluation of the code text can be migrated to ANTLR which will help pass the data and identify each keyword/statement. With this integration and further fine tuning of the code, machine earning can be integrated to the expert system so that more complex data structures and constructs can be passed as processed with speed and accuracy.

4.4. Java-ML Integration & Classification

Once the progress code is tokenized and analysed the next step is to generate the output flow diagram matching the token type of the program logic. In order to aid the decision making to identify the shape of the object that the token type logic needs to be included in the output diagram supervised classification algorithms in the Java Machine Learning Library is used.

The training dataset is first generated along with key token types with labels for the shape of the graph object it should take defined. The tokenized data from the currently running program is then added to the list for the classification algorithm to predict the shape. These predicted results are then used to draw up the diagram using JGraphX components.

4.4.1. Dataset Generation

Once the syntax tree is generated, the information can be constructed in multiple ways in order to apply Machine Learning techniques to improve the evaluations as well as to provide a more accurate and visually better output.

One simple experiment to prove the use of Machine Learning for programming language processing is to construct a prediction system for the output format for each token type of the syntax tree. 4GL language contains large number of key words and most have a pattern that when used in combination with other keywords would provide a different functionality. For example, the "FOR" keyword when used with the keyword "EACH" would indicate a table looping and when used with the keyword "FIRST" or "LAST" would indicate a fetch of a single record similar to the "FIND" keyword.

Therefore, in the experiment used for the proof of concept is to use a Supervised Learning for classification of the 4GL code syntax token type's functionality format in order to draw the output flow diagram. The training data is created by processing the sample code provided in Appendix A. The data file format would contain three attributes to begin with. The attributes considered are, the syntax token parent, the first child of the parent token node, next child of the parent token node and the class label.

The class labels for the experiment would simply be, default, loop or rhombus to use in the insertVertex method of mxGraph when drawing the output to indicate the program what functionality a certain keyword should have.

Once sample training dataset is created (Figure 4.16), this can be used along with the Machine Learning algorithms in the Java-ML library to predict the functionality of an unknown syntax detected. The experiment conducted is simple and direct and

created with the purpose of showing how AI can be used in many of ways to improve the concept of the research project. Once proved successful, this can be improved and build upon to create more advanced and eventually smarter 4GL Programming Language processing system. Further, the concept can be extracted out to be used with any programming language once the proper tokenization and training data generation is done.

4GLkeywords.csv

```
token_type_parent,token_type_child1,token_type_child2,mxgraph_vertex_shape
DISPLAY,,,default
DEFINE,VARIABLE,,default
DEFINE,TEMP-TABLE,,default
DEFINE,BUFFER,,default
DO,,,default
DO,FOR,FIRST,default
FOR,FIRST,,default
FOR,LAST,,default
PRESELECT,FIRST,,default
PRESELECT,LAST,,default
PROMPT-FOR,,,default
UPDATE,,,default
ASSIGN,WHEN,,rhombus
CASE,,,rhombus
DISPLAY,WHEN,,rhombus
DO,QUERY-TUNING,,rhombus
DO,WHILE,,rhombus
IF,CAN-DO,,rhombus
PROMPT-FOR,WHEN,,rhombus
UPDATE,WHEN,,rhombus
DO,FOR,EACH,loop
DO,PRESELECT,EACH,loop
DO,PRESELECT,FIRST,loop
DO,FOR,LAST,loop
DO,PRESELECT,LAST,loop
DO,TO,,loop
FOR,EACH,,loop
FOR,FIRST,EACH,loop
FOR,LAST,EACH,loop
PRESELECT,EACH,,loop
PRESELECT,FIRST,EACH,loop
PRESELECT,LAST,EACH,loop
PROMPT-FOR,EDITING,,loop
UPDATE,EDITING,,loop
```


Figure 4.16: Sample Dataset

Sample tree structure is shown in Figure 4.16.

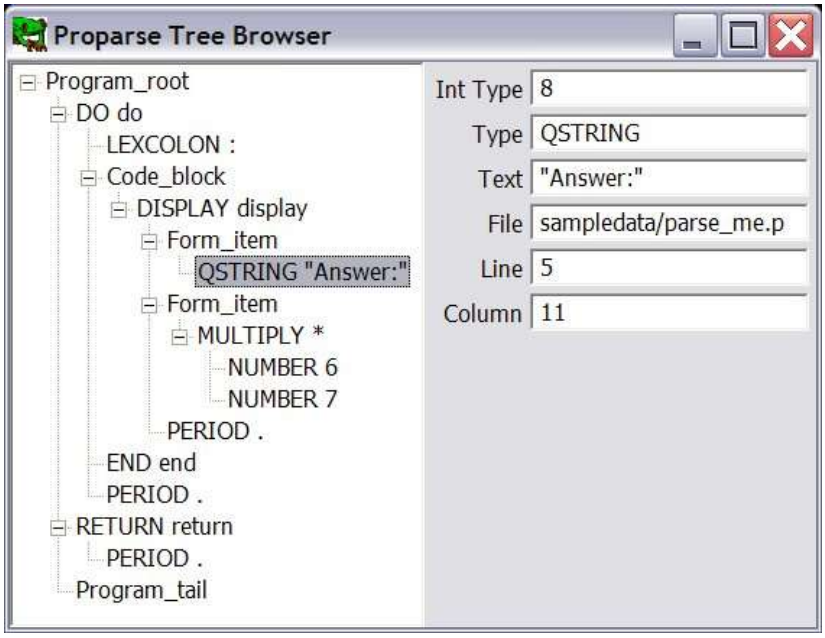


Figure 4.17: Sample Graphical View of Proparse Tree. Source [79]

4.4.2. Classification with Java-ML

```
try {
    Dataset data = FileHandler.loadDataset(new File("4GLkeywords.data"), 4, ",");
    Classifier knn = new KNearestNeighbors(5);
    knn.buildClassifier(data);
    Dataset dataForClassification = FileHandler.loadDataset(new File("4GLkeywords.data"), 4, ",");
    Map<Object, PerformanceMeasure> pm = EvaluateDataset.testDataset(knn, dataForClassification);
    for (Object o : pm.keySet())
        System.out.println(o + " - " + pm.get(o).getAccuracy());
}
```


Figure 4.18: Java-ML KNearestNeighbors

Once training dataset is prepared, using tools in Java-ML allow evaluation of the resulting dataset. With the accuracy set to an acceptable value, the predictions can be used to label the statements with the specific category. The evaluated KNearestNeghbor requires the attributes to be numeric and class label to be nominal. Once data is converted the library functions to predict can be used.

The results from the prediction is added to the respective document record in the MongoDB so that when the data is processed to generate the output flow diagram the value in the category column can be used to set the shape of the vertex.

The classification can be further extended to Cross Validate to evaluate how the statistical analysis of the dataset is generated.

4.4.3. Evaluation of the Classification for Label Prediction

In order to evaluate the classification and the prediction of label to interpret the output diagram shape by the library, WEKA 3.7 (Weikato Environment for Knowledge Analysis) is used. The evaluation techniques and results are shown in Table 4.2 with classification techniques available in WEKA against the KNearestNeighbour algorithm in the Java-ML library. The results obtained show high precision than recall and when trying to interpret the logic of the program higher precision is more important since it shows that the system gives the more relevant records precedence against the incorrect predictions. The validation is done using a test data set by altering the sample dataset in Figure 4.16 to remove few labels in as shown in Figure 4.20.

Table 4.1: Sample Dataset Attributes

Attribute	Type	Description
token_type_parent	Nominal	Beginning syntax token of progress 4GL code statement
token_type_child1	Nominal	Immediate next syntax token of the parent token of progress 4GL code statement
token_type_child2	Nominal	Immediate second syntax token of the parent token of progress 4GL code statement
Mxgraph_vertex_shape	Nominal	Class Label

Name: mxgraph_vertex_shape Type: Nominal
Missing: 0 (0%) Distinct: 3 Unique: 0 (0%)

No.	Label	Count	Weight
1	rhombus	8	8.0
2	default	11	11.0
3	loop	15	15.0

Figure 4.19: Class Labels in Training Dataset

Table 4.2: Classification Prediction Evaluation Results

Classifier Rule \ Results	Correctly Classified Instances	Incorrectly Classified Instances	Precision	Recall
NNge	76.4706%	23.5294%	0.837	0.727
Naïve Bayes Simple	73.5294%	26.4706%	0.839	0.735
J48	70.5882%	29.4118%	0.717	0.706
KNearestNeighbour	84.3137%	15.6863%	0.848	0.680

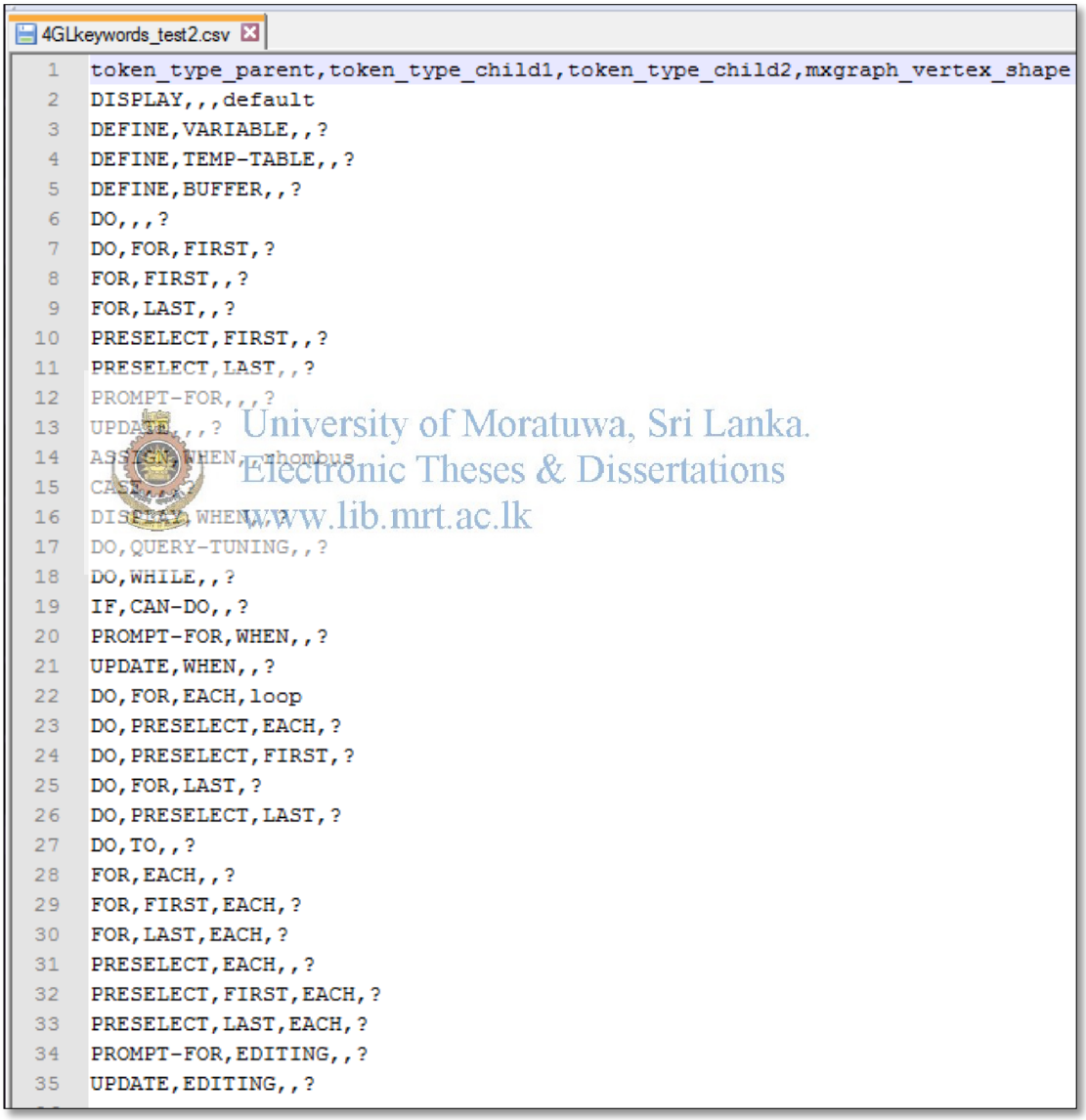


Figure 4.20: Test dataset

4.4.4. Other tools for Classification

WEKA and Rapid Minor are two popular tools for Machine Learning and both provide java libraries that can be used in programs to not only for classification but other machine learning algorithms as well [80, 81].

```
// Init RapidMiner
RapidMiner.setExecutionMode(ExecutionMode.COMMAND_LINE);
RapidMiner.init();

// Load process
final com.rapidminer.Process process =
  new com.rapidminer.Process(new File(processPath));

// Load learned model
final RepositoryLocation locWordList = new RepositoryLocation(
  "//My Machine Learning Repo/02-text-processdata/20-newsgroups.model");
final IOObject wordlist = ((IOObjectEntry)
  locWordList.locateEntry()).retrieveData(null);

// Load Wordlist
final RepositoryLocation locModel = new RepositoryLocation(
  "//My Machine Learning Repo/02-text-processdata/20-newsgroups.wordlist");
final IOObject model = ((IOObjectEntry)
  locModel.locateEntry()).retrieveData(null);

// Execute classification process with model and wordlist as
// input. It additionally expects files in
// /machine_learning/data/03-20_newsgroup_java_in
final IOContainer ioInput = new IOContainer(new IOObject[] { wordlist, model });
process.run(ioInput);
process.run(ioInput);
final long start = System.currentTimeMillis();
final IOContainer ioResult = process.run(ioInput);
final long end = System.currentTimeMillis();
System.out.println("T:" + (end - start));

// Print some results
final SimpleExampleSet ses = ioResult.get(SimpleExampleSet.class);
for (int i = 0; i &lt; Math.min(5, ses.size()); i++) {
        final Example example = ses.getExample(i);
        final Attributes attributes = example.getAttributes();

        final String id = example.getValueAsString(attributes.getId());
        final String prediction = example.getValueAsString(
          attributes.getPredictedLabel());

        System.out.println("Path: " + id + ":\tPrediction:" + prediction);
}
```


Figure 4.21: RapidMiner - Java ML Classification. Source [82]

```java
public static Evaluation classify(Classifier model,
                Instances trainingSet, Instances testingSet) throws Exception {
        Evaluation evaluation = new Evaluation(trainingSet);

        model.buildClassifier(trainingSet);
        evaluation.evaluateModel(model, testingSet);

        return evaluation;
}

public static double calculateAccuracy(FastVector predictions) {
        double correct = 0;

        for (int i = 0; i < predictions.size(); i++) {
                NominalPrediction np = (NominalPrediction) predictions.elementAt(i);
                if (np.predicted() == np.actual()) {
                        correct++;
                }
        }

        return 100 * correct / predictions.size();
}

public static Instances[][] crossValidationSplit(Instances data, int numberOfFolds) {
        Instances[][] split = new Instances[2][numberOfFolds];

        for (int i = 0; i < numberOfFolds; i++) {
                split[0][i] = data.trainCV(numberOfFolds, i);
                split[1][i] = data.testCV(numberOfFolds, i);
        }

        return split;
}
```

```java
public static void main(String[] args) throws Exception {
        BufferedReader datafile = readDataFile("weather.txt");

        Instances data = new Instances(datafile);
        data.setClassIndex(data.numAttributes() - 1);

        // Do 10-split cross validation
        Instances[][] split = crossValidationSplit(data, 10);

        // Separate split into training and testing arrays
        Instances[] trainingSplits = split[0];
        Instances[] testingSplits = split[1];

        // Use a set of classifiers
        Classifier[] models = {
                        new J48(), // a decision tree
                        new PART(),
                        new DecisionTable(),//decision table majority classifier
                        new DecisionStump() //one-level decision tree
        };

        // Run for each model
        for (int j = 0; j < models.length; j++) {

                // Collect every group of predictions for current model in a FastVector
                FastVector predictions = new FastVector();

                // For each training-testing split pair, train and test the classifier
                for (int i = 0; i < trainingSplits.length; i++) {
                        Evaluation validation = classify(models[j], trainingSplits[i], testingSplits[

                        predictions.appendElements(validation.predictions());

                        // Uncomment to see the summary for each training-testing pair.
                        //System.out.println(models[j].toString());
                }

                // Calculate overall accuracy of current classifier on all splits
                double accuracy = calculateAccuracy(predictions);

                // Print current classifier's name and accuracy in a complicated,
                // but nice-looking way.
                System.out.println("Accuracy of " + models[j].getClass().getSimpleName() + ": "
                                + String.format("%.2f%%", accuracy)
                                + "\n---------------------------------");
        }
```


Figure 4.22: Weka - Java ML Example Source [83]

4.5. Interface

The interface developed is a simple Graphical User Interface using Java JFrame class. The purpose of the research is to analyse and ensure that the proposed research is possible. Therefore focus on the interface is merely to get the user input and provide the means to notify the user of any errors/notifications regarding the file being processed.

Two versions of the interfaces are created one to type in code statements (Figure 4.23) and the other to provide a file chooser option for the program to read the code from the program file itself (Figure 4.24). These interfaces can of course be improved and enhanced with more user friendly components as future work for commercial use.

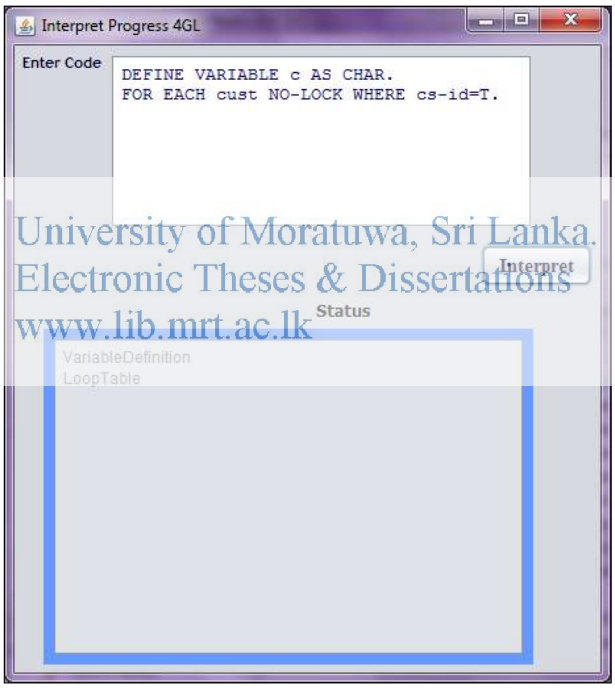


Figure 4.23: UI Interface 1

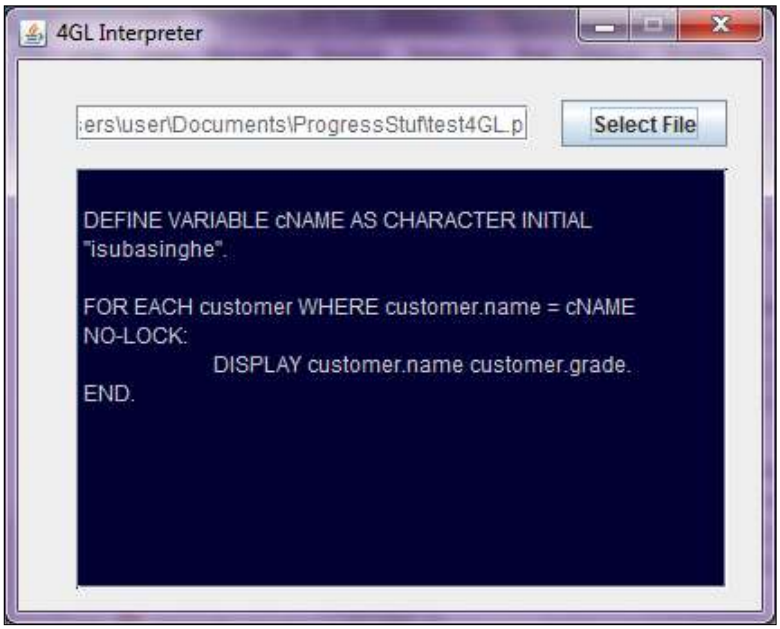


Figure 4.24: UI Interface 2 - Input File

4.6. Output Display

The proposed system is to evaluate the progress code and extract the logic to display the program workflow as flow chart visualization. For the reasons described in Section 3.4 Tools & Techniques, mxGraphs library which uses swing JGraph components is used.


```
package pkg4glinterpreter;

import com.mxgraph.swing.mxGraphComponent;
import com.mxgraph.view.mxGraph;

import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.Mongo;
```

Figure 4.25: mxGraph Integration

Instance of the mxGraph is created and initiated with the default root/parent node and the begin update method is called to start the graphic generation. Since the data for the output is store with the program code in the MongoDB collection, the DB instance is created and accessed to loop through to create the flow chart nodes.

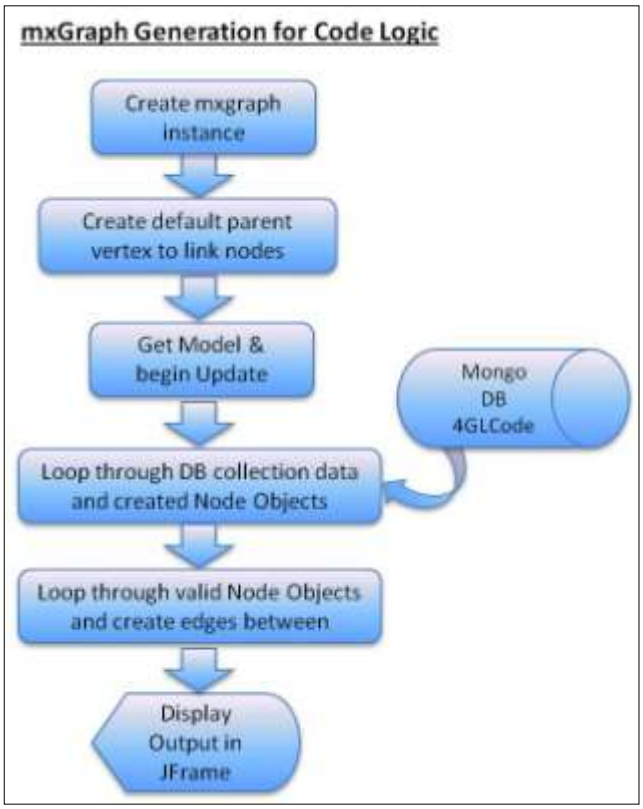


Figure 4.26: mxGraph Generation Workflow

Since there can be blank lines and invalid data, the only the valid node objects are first created using a dynamic ArrayList for mxGraph Object instances as shown in Figure 4.27.

```
while (cursor2.hasNext()) {
    // Output evaluation into flow chart
    String sEval = cursor2.next().get("eval").toString();
    if (!sEval.isEmpty() && !sEval.startsWith("Syntax")) {
        iLineCount = iLineCount + 1;
        v.add(graph.insertVertex(defaultParent, null, sEval, 20, ((j+iLineCount)*20)/2, 120, 30));
        j = j + 10;
    }
}
```

Figure 4.27: mxGraph Create Vertex

Once all nodes are created, the Array List of objects created is looped to insert the edge between nodes as shown in Figure 4.28.

```
for (int i = 0; i < iLineCount; i++) {
    if (i == (iLineCount - 1)) break;
    graph.insertEdge(defaultParent, null, "", v.get(i), v.get(i+1));
}
```

Figure 4.28: mxGraph Insert Edges

Sample output for the code provided in Figure 4.24 are shown in Figure 4.29.

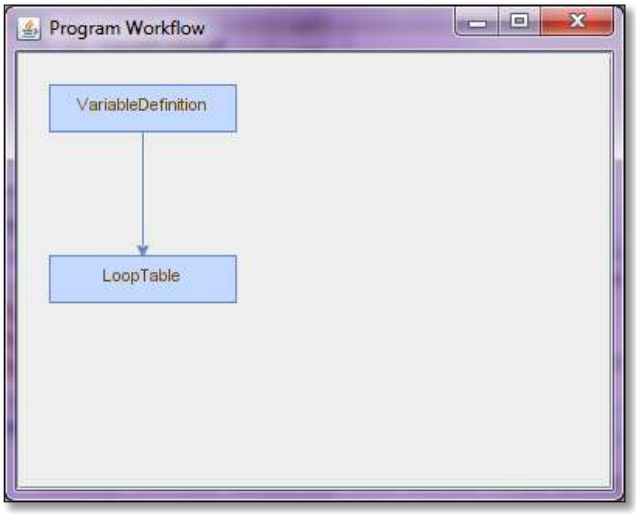


Figure 4.29: Sample Output (1)

When drawing the output using mxgraph, the shape to draw is decided by the prediction and training set results. For example, Variable Definition and Table loop are displayed in the default rectangular shape and for conditional syntax the rhombus shape is used. The below Figure 4.30 displays a sample generated for such program with IF conditional statements. More of the generated programs are included in the Appendix A section.

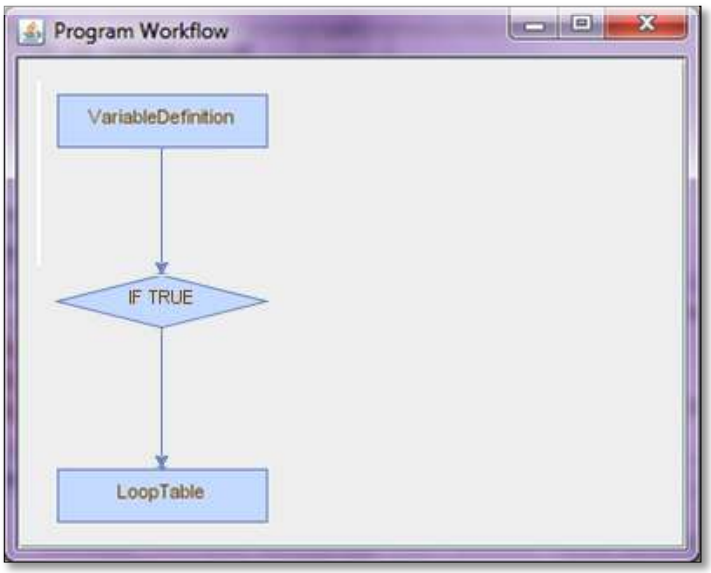


Figure 4.30: Sample Output (2)

4.7. Limitations of the Current System & Future Work

This thesis depicts the details of the research on applying Machine Learning for processing a legacy 4GL programming language to extract it code constructs and the implementation details of the prototype system to envision the idea enclosed in achieving the goal of the research. Therefore, although the methodology is in alignment with achieving the goals there are limitations to address and overcome to enhance the research area.

One such limitation is that output flow chart display window length has a limitation and larger programs are not entirely displayed. This however is a limitation of the GUI which can be overcome by breaking up the output into multiple windows and use connectors to link the program flow.

Another limitation is that the logic programming section together with the machine learning predictor is currently only depicts a sequential workflow. The programming to identify code blocks and loops are not considered and only the main sequential program logic is displayed. This section would require more of CLIPS programming as well as more of machine learning techniques can be used to expand the system to be smarter in order to display outputs for more complex programs in detail.

Furthermore, it would be useful to users if the program has an option to save the output flow as an image. These options can be integrated to the system to enhance and provide a better and more user friendly version of the application.

5. CONCLUSION

There are many expert systems today providing functionalities to complete daily business process with less cost and effort. Natural Language Processing (NLP) systems are more commonly known and vastly researched area into creating machines/systems that can not only translate between languages but also attempt to understand dialect and provide a response to match just as humans. In other words, research into creating smart systems is evolving with use of much technological advancement. Deep Learning concept is one popular area where computer scientists are attempting to create a smart system that mirrors the way a child would learn word by word and gradually learn to make sentences and understand. However, these processes require much processing power when training neural networks. High Level Programming Languages such as Fourth Generation Languages (4GL) or Artificial Languages can be viewed as a subset of Natural Languages since they are more close to the English Language. Both languages are means of communication that has its own syntax and semantics to define them individually. When these possibilities are eminent, application of same or similar tactics of Natural Language Processing (NLP) to interpret 4GL code could aid in many Programming language processing needs in the business community.

5.1. Problem with legacy systems and migration

There are many multimillion dollar organizations that make use of Fourth Generation Languages (4GL) to build their personalized business architecture and as time passes and more and more cheap technologies come into the market, migration becomes more commendable than the cost of upgrading and maintaining their current proprietary environments. However, there is a certain risk in migrating to a new environment due to the organization's massive domain knowledge which is captured in the billions of lines of code in the program files written in 4GL as well as the lack of 4GL programming knowledge experts which makes the task impractical.

There are many number of proprietary software available to translate 4GL code to other open source languages such as JAVA but using such software puts the

organization at the risk of sharing sensitive business information to a third party source. This particular problem has been addressed by many researches giving solutions from reverse engineer 4GL systems to methodical step by step translation of legacy systems. However, none provide a common solution that can be applied to any 4GL language legacy systems and to be used without fearing any sensitive data leaks.

Therefore, as described in previous chapters, the proposed research is to provide a Proof of Concept of a methodology that can be helpful for organizations looking for systems migration without having to use any additional expertise and resources while protecting the organizational data and finances. In the technological work where research concerning modelling human thinking into machines are popular, a system that is capable of learning a programming language to describe the workflow or even to translate the code would be ideal to solve the problem at hand. Therefore, the research is a proof of concept of the Programming Language Processing approach with use of Artificial Intelligent (AI) machine learning technology to support organizations to shift platforms.

5.2. Application of Machine Learning for Extracting Programming Language Constructs from 4GL Legacy Code

In order to achieve the aforementioned objectives, the proposed application is developed with the capability of reading 4GL code, separating the logic from the code statements and providing the results in a visualization of a Flow Chart.

The proposed expert system is to model a system that can model a particular workflow. The three key areas to any expert system are:

1. Fact List
2. Knowledge Base and
3. Inference Engine.

In order to provide a Graphical User Interface to gather the user input and create the necessary logic programming as well as to display the interpreted results back to the user, the application is developed in JAVA where different technologies providing different functionalities can be integrated to complete the entire system.

The logic programming tool CLIPS is the core of the developed application where the facts formats and the rules are defined. The CLIPSJNI allows the inference to be done within the project run time. In order to store the results and access the data after processing is completed, a NoSQL database MongoDB is chosen. This allows the system to freely tokenize and manipulate the data to provide results with less time and effort.

While the initial methodology for the system is proved possible and successful, the 4GL language syntax and construct complexity makes the system much more difficult to code in every line. Therefore, the proposal is to use a lexical parser that can contain the code matching logic. There are several applications that can be integrated into a Java project such as ANTLR (Another Tool for Language Recognition). For the research area in discussion the open source java library, Proparse specific for parsing Progress 4GL code is used.

With the program code is tokenized, machine learning techniques and knowledge discovery techniques can be integrated and applied to the expert system for better processing of 4GL programming language to enhance the interpretation and display of output of its core logic will be possible.

Figure 5.1 shows the well-known overview of the Knowledge Discovery Process in Databases. The research under discussion, to apply Artificial Intelligent techniques to process programming language logic, mirrors the similar steps making it possible to expand the research into achieving much more in either to infer or to extract and output essence of the program or even to translate between languages.

The Data in Figure 5.1 maps to the program code input to the proposed system. Then the next step of Pre-processing is the tokenization of the program using the Progress 4GL parser to prepare the code to be evaluated. Transformation of the data maps to the creation of the test data file which is used for to discover patterns to achieve a specific output. The results are then interpreted and evaluated and based on the success rate used to derive the final output.

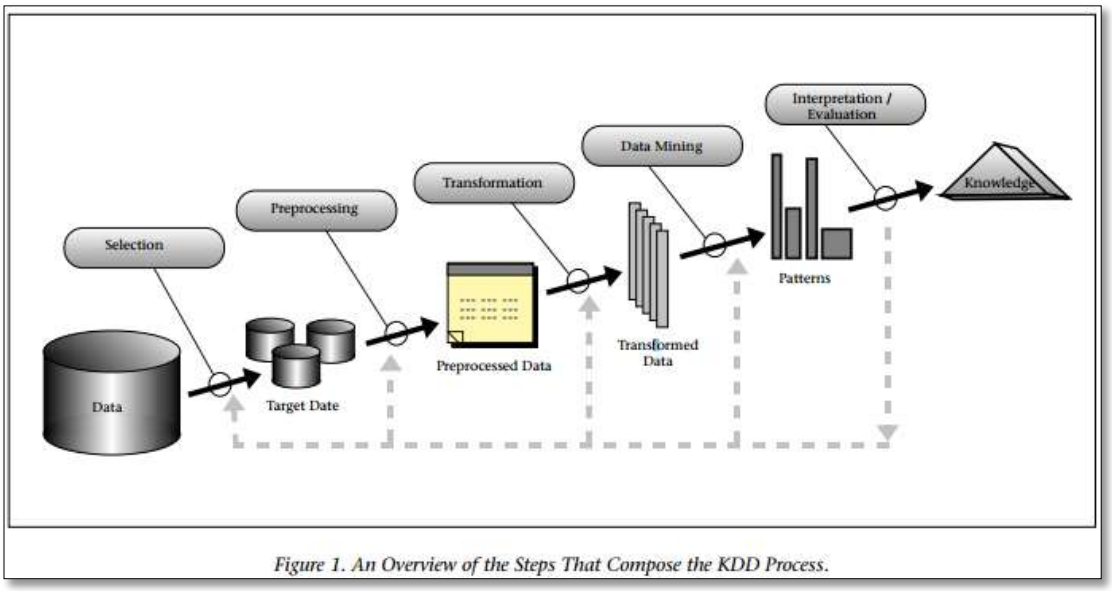


Figure 5.1: Knowledge Discovery Process Overview. Source [84]

5.3. Summary

In summary, the research puts together different technologies to work in one program in order to interpret a 4GL code and infer the response using the CLIPS logic programming tool and output the results in a graphical representation using mxGraphs. The essence of Artificial Intelligence to support the research objectives is gained by Java-ML Machine Learning library. This project merely opens a door to an area of research where Machine Learning can be used for various interpretations of the processing programming languages.

In theory, similar to Natural Language Processing, Machine Learning through Decision Tree analysis and building Expert Systems using AI concepts can be applied to process rules and relationships of Programming Languages. However, Programming Languages contains complex logic that put together key words to form special instructions and two languages are not similar and even with rules are extracted from the languages mapping the two to convert one to another would therefore be a complex task.

As future work for the 4GL Interpreter, the interpreted logic can be mapped to a different programming language such as JAVA and allow translation from Progress 4GL to Java. However, in order to provide full translation of the program, the

transformation program needs to have equal or more capabilities than Progress® 4GL if not some functionalities may be left incomplete. The idea is that, most programs in Progress® are sequential, and therefore an Object Oriented language such as Java could cover most of the language constructs and to cover the database interactions, Java with Hibernate can be used.

REFERENCES

[1]. Progress 4GL - fourth generation language. (2014). *What is Progress 4GL?* [Online]. Available: http://www.progress4gl.com/index.html. [Accessed: February 23, 2015].

[2]. Progress Software. (2015). *Who We Are - History* [Online]. Available: https://www.progress.com/company-info/who-we-are/history. [Accessed: February 23, 2015].

[3]. Progress Software. (2013, August 17). *4GL Languages* [Online]. Available: http://www.progress.com/en/4gl-languages.html. [Accessed: February 25, 2015].

[4]. Progress Software. (2013, August 17). *Business Application Integration* [Online]. Available: http://www.progress.com/en/business-application-integration.html. [Accessed: February 25, 2015].

[5]. Wikipedia. (2004, February 6). *Fourth-generation programming language* [Online]. Available: http://en.wikipedia.org/wiki/Fourth-generation_programming_language. [Accessed: March 07, 2015].

[6]. IBM Software. (2014). IBM® Informix® 4GL [Online]. Available: http://www-03.ibm.com/software/products/en/4gl/. [Accessed: February 2, 2014].

[7]. Progress Software. (2014). *Progress Software* [Online] Available: http://progress4gl.com/. [Accessed: June 30, 2014].

[8]. IBM Software, “Transitioning: Informix 4GL To Enterprise Generation Language EGL” in *IBM Press*, 2005.

[9]. C. Nagy, L. Vidács, R. Ferenc, T. Gyimóthy, F. Kocsis, and I. Kovács. "Solutions for reverse engineering 4GL applications, recovering the design of a logistical wholesale system." in *Software Maintenance and Reengineering (CSMR)*, 2011 15th European Conference on, pp. 343-346. IEEE, 2011.

[10]. M. Rahgozar and F. Oroumchian. "A Transformational Approach for Legacy Systems’ evolution." in *Submitted for publication in: 2002 WSEAS International Conference on Applied Mathematics and Computer Science (AMCOS’02)*, Copacabana, Rio De Janeiro. 2002.

[11]. B. Wilson and T. Van der Beken. "Observations on automation in cross-platform migration." in *ELISA workshop*, p. 140.

[12]. J. C. Riquelme, M. Polo, J. S. Aguilar–Ruiz, M. Piattini, F. J. Ferrer–Troyano, and F. Ruiz. "A comparison of effort estimation methods for 4GL

programs: experiences with Statistics and Data Mining." in *International Journal of Software Engineering and Knowledge Engineering* 16, no. 01, pp. 127-140, 2006.

[13]. Ispirer Systems Ltd. (1999-2014). *Migrate Informix to Oracle* [Online]. Available: http://www.ispirer.com/products/informix-to-oracle-migration. [Accessed: June 30, 2014].

[14]. IBM Information Management Software. (2014). *Converting IBM Informix 4GL Applications to Informix Genero* [Online] Available: ftp://public.dhe.ibm.com/software/data/sw-library/informix/whitepapers/I4GL-to-Genero-Conversion.pdf. [Accessed: June 30, 2014].

[15]. IBM Software. (2011, March). *Infomix Genero* [Online]. Available: http://www-03.ibm.com/software/products/en/infogene/. [Accessed: June 30, 2014].

[16]. MoreData. *The 4GL Connector White Paper*. Informix an IBM trademark 2001-2014.

[17]. MoreData. (2001-2014). *INFORMIX specialized services* [Online]. Available: http://www.moredata.eu/offer/informix.html. [Accessed: June 30, 2014].

[18]. FreeSoft Inc. (2012). *Legacy Modernization* [Online]. Available: http://www.freesoftus.com/en/solutions/legacy_modernization/. [Accessed: June 30, 2014].

[19]. R. Nicola, "Conversion to SAS Software from other 4GL Languages", *Quality Partner Forum*, SPS Software Services Inc. Canton, Ohio.

[20]. Raincode SPRL. "Automatic migration of your Progress 4GL application to Java", *in EU Gateway Program*, Belgium.

[21]. Global Code Development Corporation, "Progress 4GL to Java Conversion – For Independent Software Vendors", *for Golden Code Development Corporation Documentation*, 2006-2007.

[22]. J. V. Harrison, and W. M. Lim. "Automated reverse engineering of legacy 4GL information system applications using the ITOC workbench." in *Advanced Information Systems Engineering*, pp. 41-57. Springer Berlin Heidelberg, 1998.

[23]. C. Nagy, L. Vidács, R. Ferenc, T. Gyimóthy, F. Kocsis, and I. Kovács. "Solutions for reverse engineering 4GL applications, recovering the design of a logistical wholesale system." in *Software Maintenance and Reengineering (CSMR)*, 2011 15th European Conference on, pp. 343-346. IEEE, 2011.

[24]. M. Rahgozar and F. Oroumchian. "A Transformational Approach for Legacy Systems' evolution." in *Submitted for publication in: 2002 WSEAS International Conference on Applied Mathematics and Computer Science (AMCOS'02)*, Copacabana, Rio De Janeiro. 2002.

[25]. B. Wilson and T. Van der Beken. "Observations on automation in cross-platform migration." in *ELISA workshop*, pp. 140.

[26]. R. Ferenc, Á. Beszédes, M. Tarkiainen, and T. Gyimóthy, "Columbus–Reverse Engineering Tool and Schema for C++," in *Proceedings of the 18th International Conference on Software Maintenance (ICSM'02)*.IEEE Computer Society, pp. 172–181, 2002.

[27]. J. C. Riquelme, M. Polo, J. S. Aguilar–Ruiz, M. Piattini, F. J. Ferrer–Troyano, and F. Ruiz. "A comparison of effort estimation methods for 4GL programs: experiences with Statistics and Data Mining." in *International Journal of Software Engineering and Knowledge Engineering* 16, no. 01, pp. 127-140, 2006.

[28]. K. K. Aggarwal, Y. Singh, P. Chandra, and M. Puri. "Bayesian regularization in a neural network model to estimate lines of code using function points." in *Journal of Computer Science* 1, no. 4, pp. 505, 2005.

[29]. L. L. Minku and X. Yao. "A principled evaluation of ensembles of learning machines for software effort estimation." in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, p. 9. ACM, 2011.

[30]. G. E. Wittigand G. R. Finnic. "Using artificial neural networks and function points to estimate 4GL software development effort." in *Australasian Journal of Information Systems* 1, no. 2, 2007.

[31]. Google (2010, Aug 10). *Google Translate: Find out how our translations are created.* [Online]. Available: http://translate.google.com.au/about/. [Accessed: March 07, 2015].

[32]. P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, E. Herbst (2013, Aug 13). *Moses: Statistical Machine Translation System*. [Online]. Available: http://www.statmt.org/moses/?n=Moses.Overview. [Accessed: March 07, 2015].

[33]. Google Cloud Platform (2014, October 10). *Google Prediction API*. [Online]. Available: https://cloud.google.com/prediction/docs/developer-guide

[34]. Oracle Web Determinations. *Create a new language translation for a rulebase*. [Online]. Available: http://docs.oracle.com/html/E24270_01/Content/Languages/Create_new_language_translation_for_rulebase.htm/. [Accessed: March 07, 2015].

[35]. R. D. Brown, “Adding linguistic knowledge to a lexical example-based translation system” in *Proceedings of the Eighth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-99)*, pp. 22-32. 1999.

[36]. Y. Yang, G. C. J. G. Carbonell, R. D. Brown, T. Pierce, B. T. Archibald, and X. Liu. “Learning approaches for detecting and tracking news events” in *IEEE Intelligent Systems* 14, no. 4 (1999): 32-43.

[37]. A. B. Phillips and R. D. Brown. “Cunei machine translation platform: System description.” in *3rd Workshop on Example-Based Machine Translation*. 2009.

[38]. R. D. Brown, “Example-based machine translation in the pangloss system.” in *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pp. 169-174. Association for Computational Linguistics, 1996.

[39]. A. B. Phillips and R. D. Brown. “Training machine translation with a second-order taylor approximation of weighted translation instances.” in *MT Summit*. 2011.

[40]. C. Monson, A. F. Llitjós, R. Aranovich, L. Levin, R. D. Brown, E. Peterson, J. Carbonell, and A. Lavie. “Building NLP systems for two resource-scarce indigenous languages: Mapudungun and Quechua.” *Strategies for developing machine translation for minority languages* (2006): 15.

[41]. R. Gangadharaiah, R. D. Brown, and J. Carbonell. “Spectral clustering for example based machine translation.” in *Proceedings of the Human Language Technology Conference of the NAACL*, Companion Volume: Short Papers, pp. 41-44. Association for Computational Linguistics, 2006.

[42]. Code Project – Anshul (2010, August 10). *Develop Your Own Language Translation System*. [Online]. Available: http://www.codeproject.com/Articles/100126/Develop-Your-Own-Language-Translation-System. [Accessed: March 19, 2015]

[43]. S. Eiichiro, H. Iida, and H. Kohyama. "Translating with examples: a new approach to machine translation." in *The Third International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Language*, no. 3, pp. 203-212. 1990.

[44]. P. Ciancarini,M. J. Wooldridge et.al, “Collection of Revised papers in Agent-Oriented Software Engineering” in *First International Workshop, AOSE* 2000, Limerick, Ireland, June 10, 2000

[45]. M. Simard, N. Ueffing, P. Isabelle, and R. Kuhn. “Rule-based translation with statistical phrase-based post-editing” in *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 203–206, Prague, Czech Republic, June 2007. Association for Computational Linguistics

[46]. J. McDermott. "R1: A rule-based configurer of computer systems." in *Artificial intelligence* 19, no. 1, pp. 39-88, 1982.

[47]. Hayes-Roth, Frederick. “Rule-based systems” in *Communications of the ACM* 28, no. 9, pp. 921-932, 1985.

[48]. P. F.Brown, J. Cocke, S. A. Della Pietra, V. J. Della Pietra, F. Jelinek, J. C. Lai, and R. L. Mercer. "Method and system for natural language translation." U.S. Patent 5,477,451, issued December 19, 1995.

[49]. I. Fliege, R. Grammes, and C. Weber. "ConTraST–a configurable SDL transpiler and runtime environment." in *System Analysis and Modeling: Language Profiles*, Springer Berlin Heidelberg, pp. 216-228, 2006.

[50]. P. Moreau, C. Ringeissen, and M. Vittek. "A pattern-matching compiler." In *Electronic Notes in Theoretical Computer Science* 44, no. 2, pp. 161-180, 2001.

[51]. S. I.Feldman, "A Fortran to C converter," in *ACM SIGPLAN FORTRAN Forum*, vol. 9, no. 2, pp. 21-22. ACM, 1990.

[52]. R. L.London,J. V. Guttag, J. J. Horning, B. W. Lampson, J. G. Mitchell, and G. J. Popek, *Proof rules for the programming language Euclid*, Springer Berlin Heidelberg, 1978, pp 1-26.

[53]. L.A. Hayden, "System and method for generating target language code utilizing an object oriented code generator," U.S. Patent 6,877,155, issued April 5, 2005.

[54]. J. V. Harrison, A. Berglas, and I. Peake, "Legacy 4GL application migration via knowledge-based software engineering technology: a case study," in *Software Engineering Conference*, 1997. Proceedings. 1997 Australian, pp. 70-78. IEEE, 1997.

[55]. M. Rahgozar and F. Oroumchian, "An effective strategy for legacy systems evolution," *Journal of Software Maintenance and Evolution: Research and Practice* 15, no. 5, pp. 325-344, 2003.

[56]. Bauer, C. and King, G., *Hibernate In Action, Practical Object/Relational Mapping*, Manning Publications, 2004.

[57]. Bauer, Christian, and Gavin King. *Java Persistance with Hibernate.* Dreamtech Press, 2006.

[58]. S. J. Russell, P. Norvig, J. F. Canny, Jitendra M. Malik, and Douglas D. Edwards. *Artificial intelligence: a modern approach.* Vol. 74. Englewood Cliffs: Prentice hall, 1995.

[59]. J. F. Allen, *Natural Language Processing.* [Online]. Available: http://ai.ato.ms/MITECS/Entry/allen.html. [Accessed: March 16, 2015]

[60]. R. Wolniewicz, "Auto-Coding and Natural Language Processing." in *3M Health Information Systems*, 2011.

[61]. MIT Technology Review (2013, April 23), *10 Breakthrough Technologies 2013 – Deep Learning*. [Online]. Available: http://www.technologyreview.com/featuredstory/513696/deep-learning/. [Accessed: February 27, 2015].

[62]. Terrance Parr (2014), *ANTLR – Another Tool for Language Recognition*. [Online]. Available: http://www.antlr.org/index.html. [Accessed: February 27, 2015].

[63]. Terrance Parr (2015, January 15), *Getting Started with ANTLR v4*. [Online]. Available: https://theantlrguy.atlassian.net/wiki/display/ANTLR4/Getting+Started+with+ANTLR+v4. [Accessed: February 27, 2015].

[64]. MongoDB Inc. (2015), *mongoDB - Agile and Scalable.* [Online]. Available: http://www.mongodb.org/. [Accessed: February 27, 2015].

[65]. SourceForge.NET (2015, February 1), *CLIPS – Tool for Building Expert Systems*. [Online]. Available: http://clipsrules.sourceforge.net. [Accessed: February 27, 2015].

[66]. JGraph Ltd. (2015), *mxGraph is the market leading JavaScript Graph Visualization Component*. [Online]. Available: https://www.jgraph.com/javascript-graph-visualization-library.html. [Accessed: February 27, 2015]. [Accessed: February 27, 2015].

[67]. JGraphX Version 3.1.2.2 (2015, February 09), *JGraphX (JGraph 6) User Manual*. [Online]. Available: http://jgraph.github.io/mxgraph/docs/manual_javavis.html. [Accessed: February 27, 2015].

[68]. Dr. Shazzad Hosain (2013, March 24), *CLIPS Tutorial 1*. [PDF]. Available: http://w3.northsouth.edu/php/faculty/shazzad/courses/cse513_cse348_ete333/course_reading/CLIPS%20tutorial.pdf. [Accessed: February 27, 2015].

[69]. Progress Software Corporation (1993 - 2015), *OpenEdge Documentation – Progress OpenEdge Release 11 Master Help System* [HTML]

[70]. S. R. Schach, *Object-oriented and classical software engineering*, 7th ed. New York: WCB/McGraw-Hill, 2007.

[71]. J. Brownlee (2014, July 18). *Java Machine Learning*. [Online]. Available: http://machinelearningmastery.com/java-machine-learning/

[72]. S. Yegulalp (2014, August 4). *5 ways to add machine learning to Java, JavaScript, and more*. [Online]. Available: http://www.javaworld.com/article/2461485/big-data/5-ways-to-add-machine-learning-to-java-javascript-and-more.html. [Accessed: March 07, 2015].

[73]. A. C. Oliver (2014, May 29). *Enjoy Machine Learning with Mahout on Hadoop*. [Online]. Available: http://www.infoworld.com/article/2608418/application-development/enjoy-machine-learning-with-mahout-on-hadoop.html. [Accessed: March 07, 2015].

[74]. BigML Inc. (2012, June 7). *Machine Learning in Java has never been easier!* [Online]. Available: http://blog.bigml.com/2012/06/07/machine-learning-in-java-has-never-been-easier/. [Accessed: March 08, 2015].

[75]. I. Vasilev. *A Deep Learning Tutorial: From Perceptrons to Deep Networks*. [Online]. Available: http://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks. [Accessed: March 24, 2015].

[76]. Joanju Proparse (2011, November 27). *Parse your OpenEdge ABL source code*. [Online]. Available: http://www.joanju.com/proparse/index.php. [Accessed: March 15, 2015].

[77]. Abeel, Thomas, Yves Van de Peer, and Yvan Saeys. "Java-ML: A machine learning library." in *The Journal of Machine Learning Research 10* (2009): 931-934.

[78]. J. Green, The Open Edge Hive (2009, August 24). *The Proparse Library*. [Online]. Available: http://www.oehive.org/node/1514. [Accessed: March 20, 2015].

[79]. J. Green, The Open Edge Hive (2006, September 05). *Proparse Java/SWT tree view* [Online]. Available: http://www.oehive.org/node/150. [Accessed: March 11, 2015].

[80]. University of Waikato, *Weka 3: Data Mining Software in Java* [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/. [Accessed: March 11, 2015].

[81]. RapidMiner (2014), *RapidMiner – Analytics for Anyone* [Online]. Available: https://rapidminer.com/. [Accessed: March 11, 2015].

[82]. F. Engelen (2013, March 22), *Taking a Look at Java-based Machine Lerning Classification* [Online]. Available: https://blog.codecentric.de/en/2013/03/java-based-machine-learning-by-classification/. [Accessed: March 11, 2015].

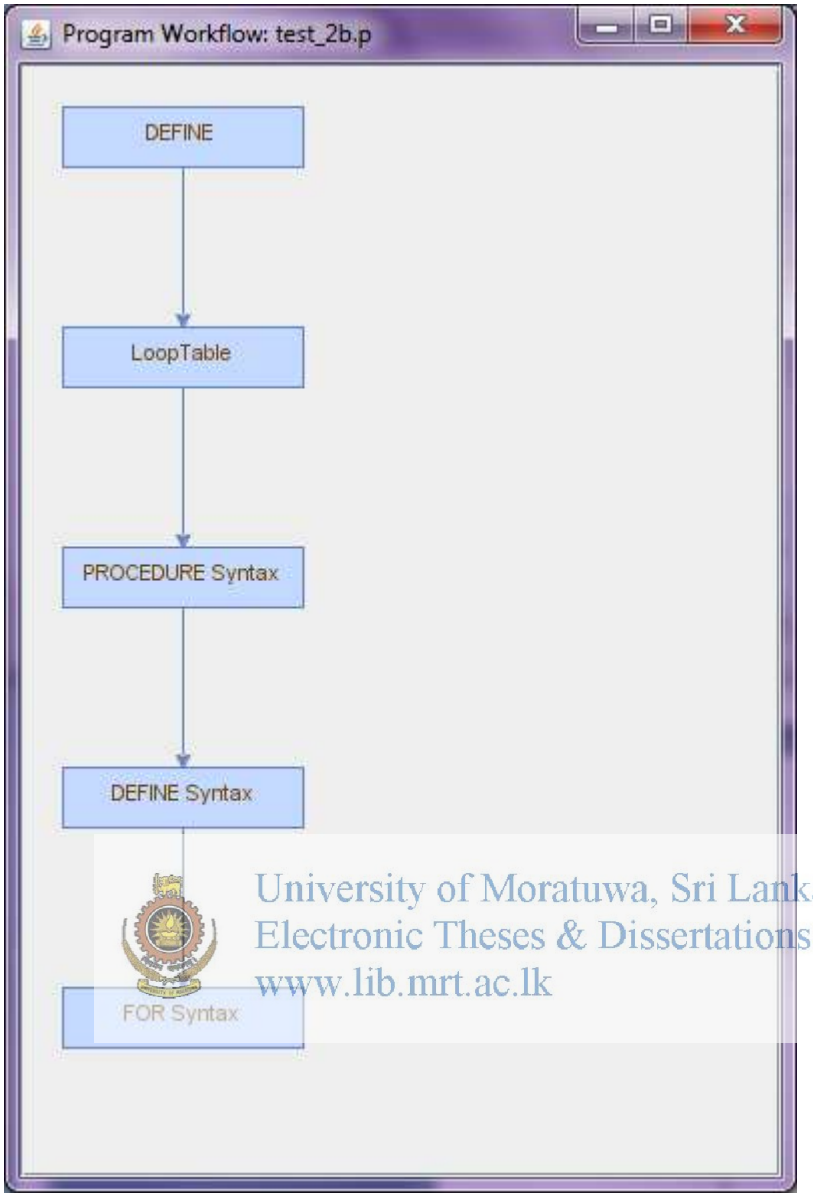
[83]. X. Wang (2013, January), *A Simple Machine Learning Example in Java* [Online]. Available: http://www.programcreek.com/2013/01/a-simple-machine-learning-example-in-java/. [Accessed: March 11, 2015].

[84]. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. "From Data Mining to Knowledge Discovery in Databases." in *AI magazine 17*, no. 3, 1996, pp. 37.

APPENDIX A: 4GL PROGRAM CODE & OUTPUT

- **Sample Program 1**

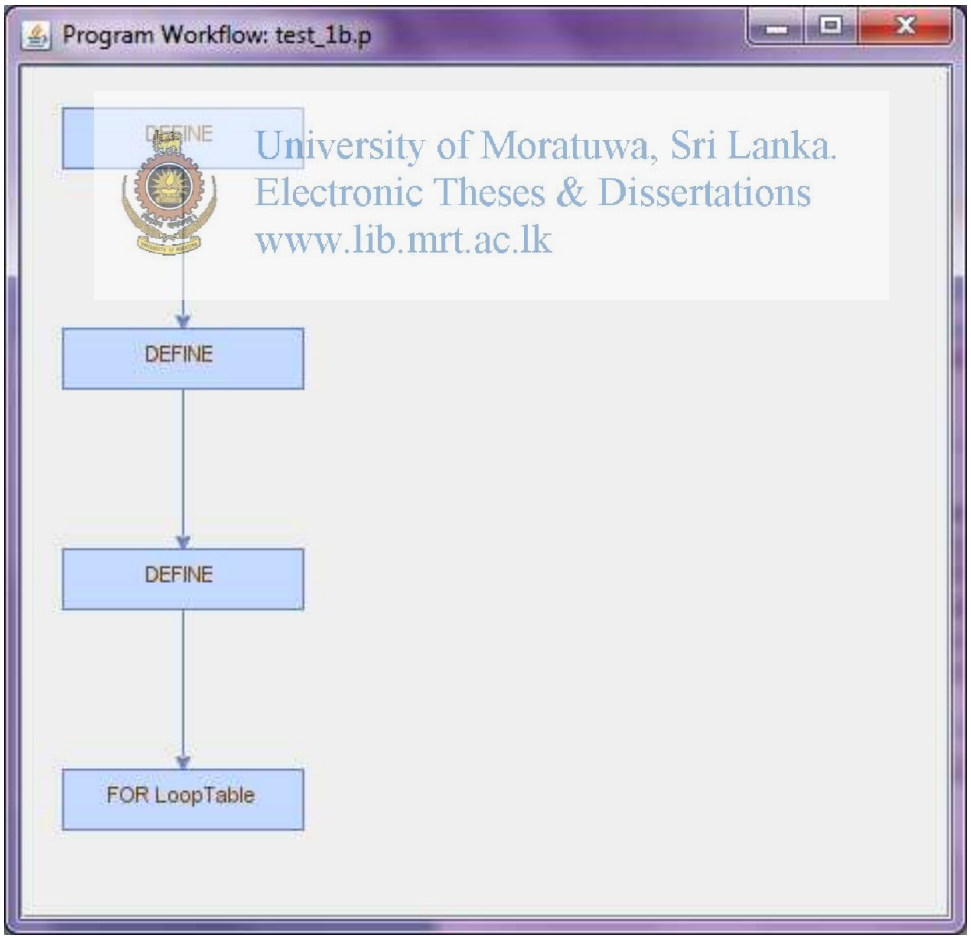
```
/*------------------------------------------------------------------
   File       : test_2b.p
   Description : Write a call to an internal procedure to calculate the number of orders the customer
placed during this year, and their total value. Display these values as part of (a) above
   Author(s)   : ISubasinghe
   Created     : Wed Mar 17 08:40:44 IST 2010
   Notes       :
 ----------------------------------------------------------------------*/

/* ***************************  Definitions  ************************** */
DEFINE VARIABLE oCount AS INTEGER LABEL "No of Orders".
DEFINE VARIABLE oTotal AS INTEGER LABEL "Total".

/* ***************************  Main Block  *************************** */
FOR EACH Customer BREAK BY Customer.CustNum:
  RUN NoOfCustOrders(INPUT customer.CustNum, OUTPUT oCount, OUTPUT oTotal).
  DISPLAY Customer.CustNum
          Customer.Name
          Customer.CreditLimit
          oCount
          oTotal
             WITH SIZE 100 BY 100 .

END.



PROCEDURE NoOfCustOrders:
   DEFINE INPUT  PARAMETER ipCustNum LIKE customer.CustNum NO-UNDO.
   DEFINE OUTPUT PARAMETER oCount AS INTEGER.
   DEFIN  OUTPUT PARAMETER oTotal AS INTEGER.

   FOR EACH ORDER
      WHERE ORDER.CustNum        = ipCustNum
       AND YEAR(Order.OrderDate) = 1997:

     FOR EACH ORDERLINE OF ORDER:
       oCount = oCount + 1.
       oTotal = oTotal + (OrderLine.Qty * OrderLine.Price).
       /*ACCUMULATE OrderLine.Qty * OrderLine.Price (TOTAL).
       DISPLAY (ACCUM TOTAL OrderLine.Qty * OrderLine.Price) LABEL "Total".*/
     END.
   END.
END.
```



- **Sample Program 2**

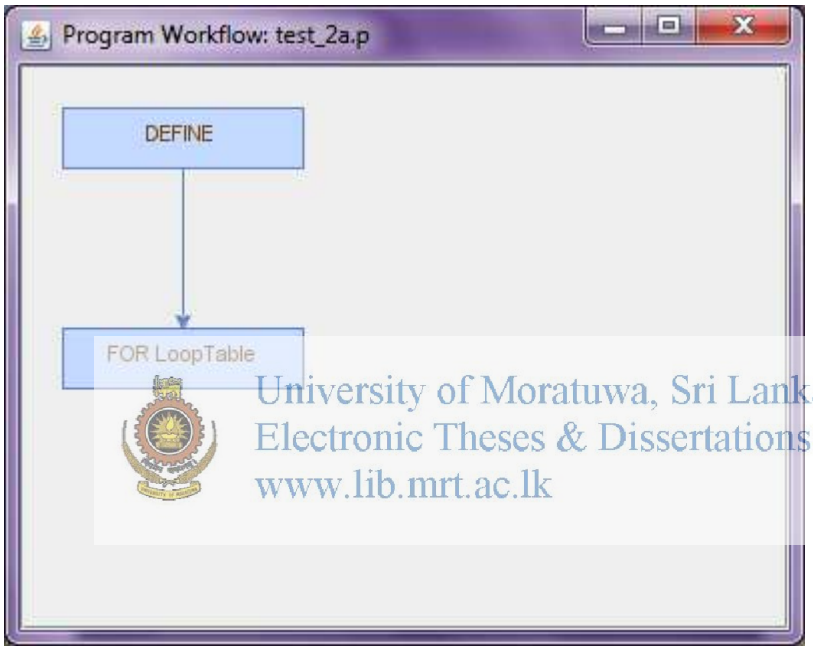
```
/*------------------------------------------------------------------------
   File        : test_1b.p
   Description : Modify the above code to delete all records where the credit limit is less than 100 and
there are no orders placed. Both tasks should be performed within the same loop.
   Author(s)   : ISubasinghe
   Created     : Tue Mar 16 18:23:23 IST 2010
   Notes       :
 ----------------------------------------------------------------------*/

/* ***************************  Definitions  ************************** */
DEFINE VARIABLE cNum   LIKE Customer.CustNum NO-UNDO.
DEFINE VARIABLE cState LIKE Customer.State    NO-UNDO.

DEFINE FRAME frmHead
```

```
    HEADER
     "Customers in " cState
     WITH NO-LABEL.


/* ************************** Main Block ************************** */

FOR EACH Customer BREAK BY Customer.State:
   IF FIRST-OF(Customer.State) THEN
         DISPLAY Customer.State WITH FRAME frmHead NO-LABEL.
   DISPLAY Customer.CustNum
         Customer.Name
         Customer.CreditLimit.

   FIND FIRST Order OF CUSTOMER NO-LOCK NO-ERROR.
   IF AVAILABLE(Order) AND Customer.CreditLimit < 20000 THEN
   DO:
      ASSIGN cNum = Customer.CustNum.
      DELETE Customer.
      MESSAGE "CUSTOMER " cNum " DELETED".

   END.
END.
```



- **Sample Program 3**

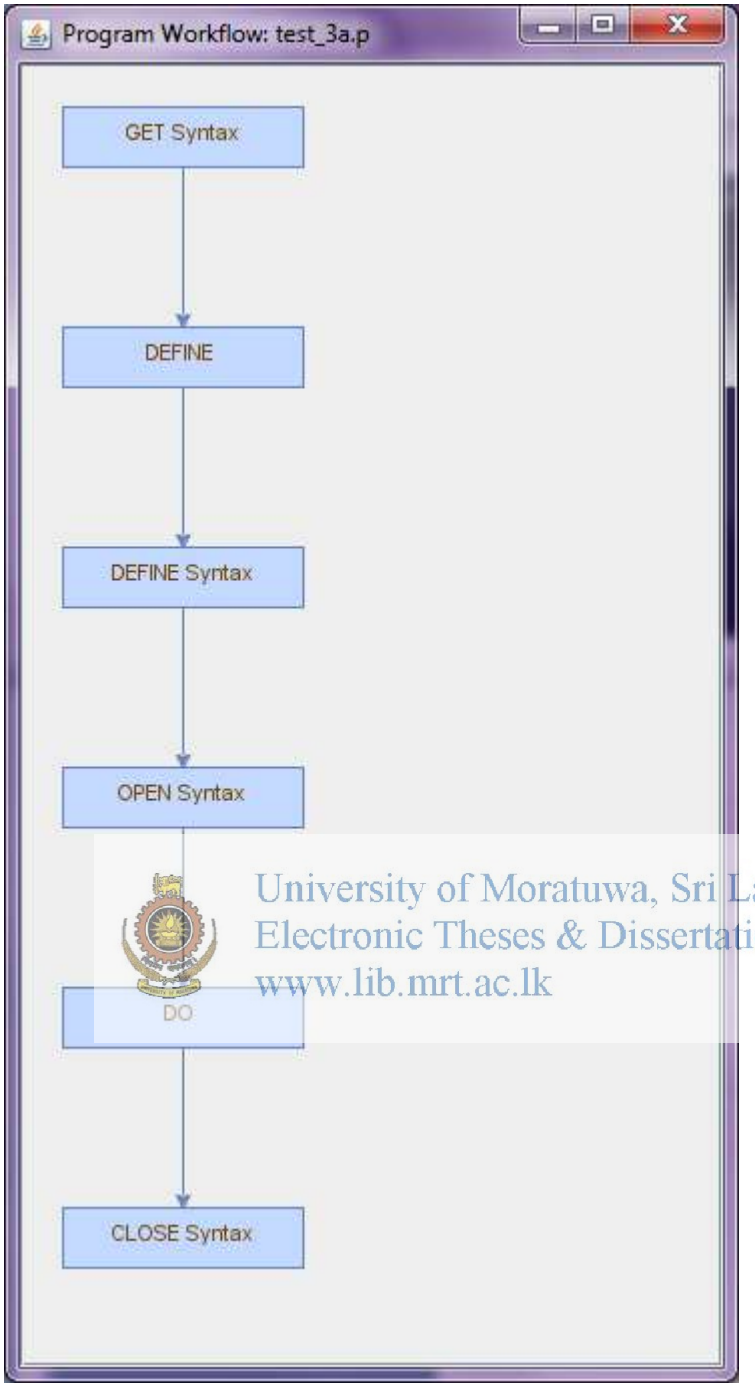
```
/*------------------------------------------------------------------
   File        : test_2a.p
   Description : Write a procedure which goes through the customer table and displays customer code,
name and credit limit.
   Author(s)   : ISubasinghe
   Created     : Wed Mar 17 08:18:32 IST 2010
   Notes       :
 ----------------------------------------------------------------------*/

/* ***************************  Main Block  *************************** */
FOR EACH Customer:
   DISPLAY Customer.CustNum
           Customer.Name
           Customer.CreditLimit.
END.
```



- **Sample Program 5**

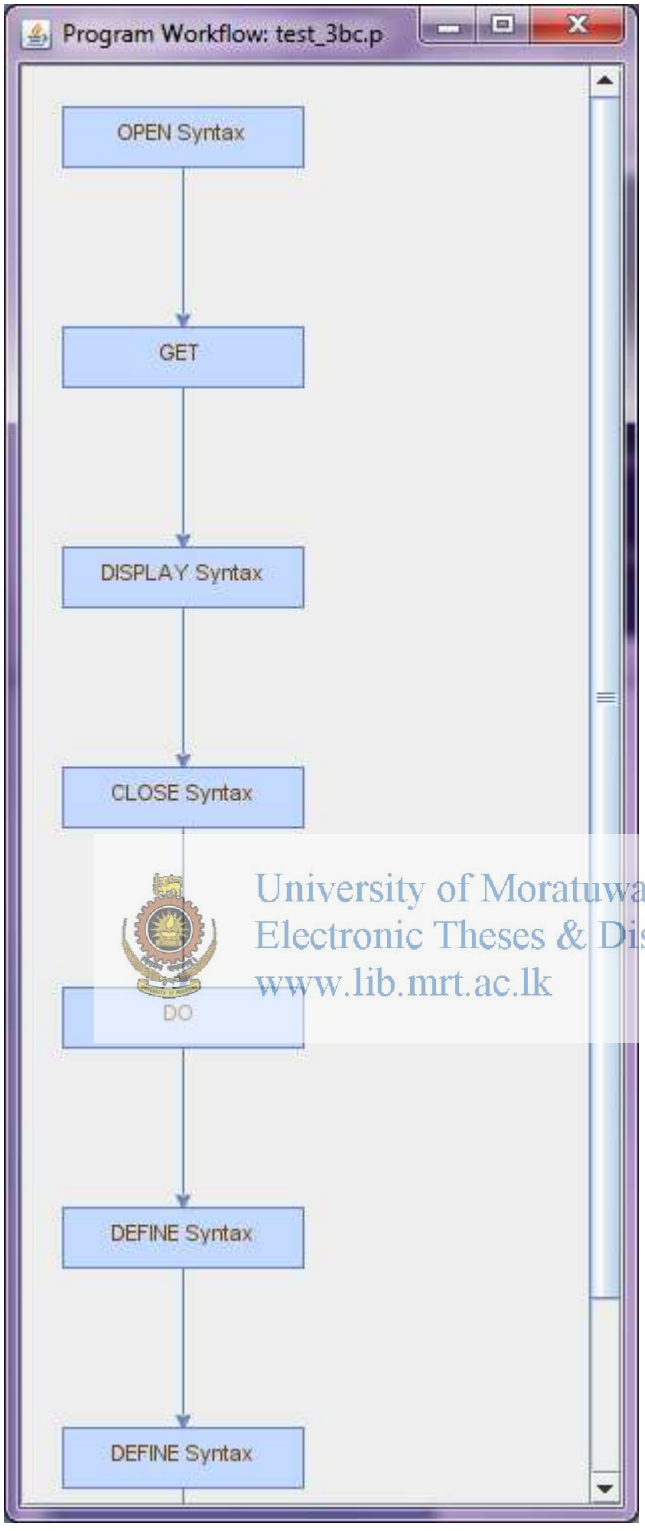
```
/*------------------------------------------------------------------
   File        : test_3a
   Description : Use a query to display the customer code, customer name, and the number of
customer results at each line.
   Author(s)   : ISubasinghe
   Created     : Wed Mar 17 10:36:19 IST 2010
   Notes       :
 ----------------------------------------------------------------------*/

/* ***************************  Definitions  ************************** */
DEFINE QUERY    q1  FOR Customer SCROLLING.
DEFINE VARIABLE cnt AS INT.

/* ***************************  Main Block  *************************** */
```

```
OPEN QUERY q1 FOR EACH Customer.

GET FIRST q1 NO-LOCK.

cnt = NUM-RESULTS ("q1").

DO WHILE NOT QUERY-OFF-END ("q1")WITH FRAME DEFAULT-FRAME:

   DISPLAY Customer.CustNum
         Customer.Name
         cnt LABEL "No Of Results".
   GET NEXT q1 NO-LOCK.
   cnt = NUM-RESULTS ("q1").
END.
CLOSE QUERY q1.
```



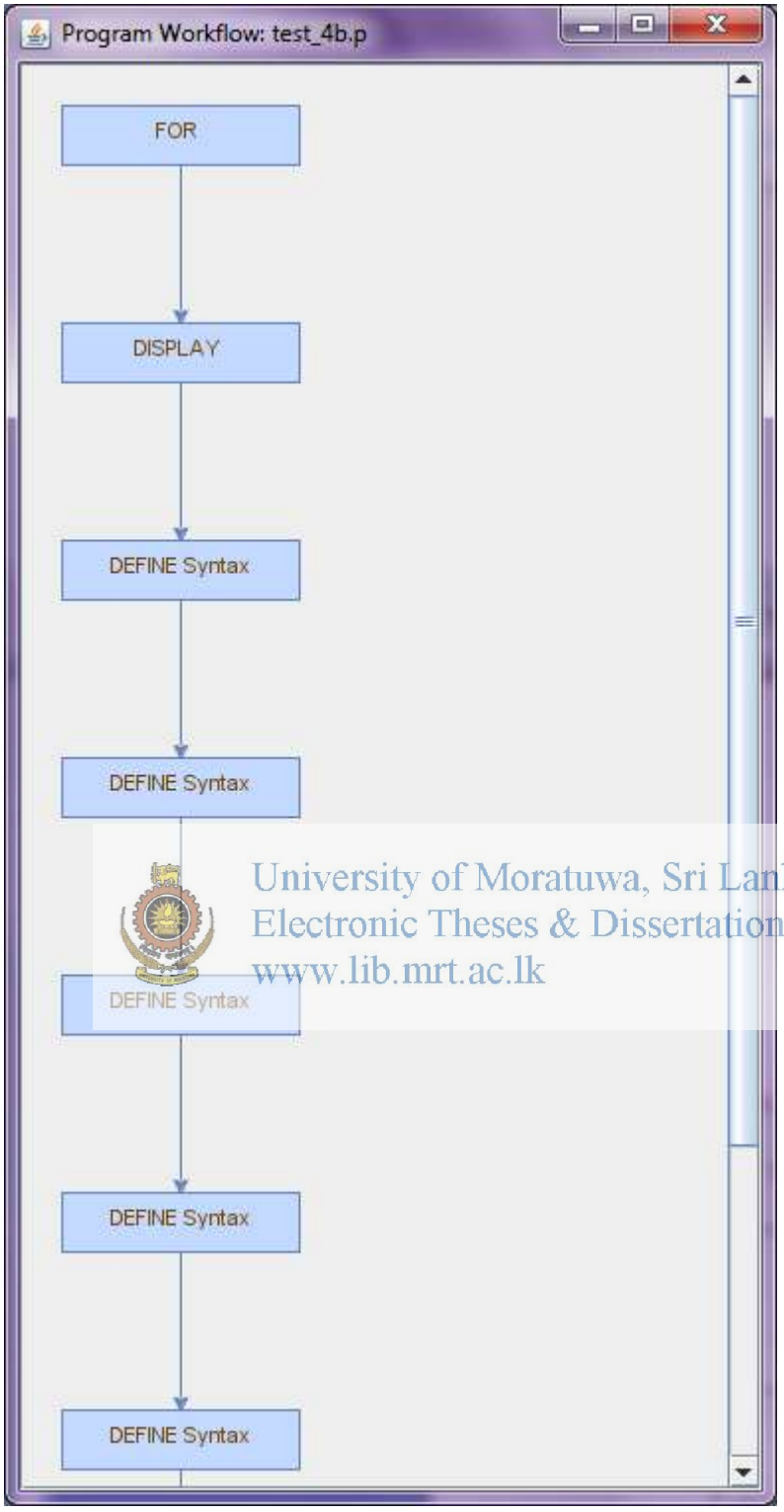
- **Sample Program 6**

```
/*------------------------------------------------------------------------
   File        : test_3b.p
   Description : b. Define and open a query that will preselect all customers, and all the orders for each
customer, if available.
                 c. Display the customer code and name, the number of orders available per customer, and
the value of these orders.
   Author(s)   : ISubasinghe
   Created     : Wed Mar 17 11:45:08 IST 2010
   Notes       :
 ----------------------------------------------------------------------*/

/* ***************************  Definitions  ************************** */
DEFINE QUERY q1 FOR Customer, Order SCROLLING.

DEFINE VARIABLE cNum   LIKE Customer.CustNum.
DEFINE VARIABLE oCount AS   INT.

/* ***************************  Main Block  *************************** */
OPEN QUERY q1
    PRESELECT EACH Customer, EACH Order OF Customer.

DISPLAY NUM-RESULTS ("q1") LABEL "No Of Results" WITH FRAME frmHdr.

GET FIRST q1 NO-LOCK.

DO WHILE NOT QUERY-OFF-END ("q1") WITH FRAME DEFAULT-FRAME:

  ASSIGN cNum = Customer.CustNum.
  MESSAGE cNum VIEW-AS ALERT-BOX.
   DO WHILE Customer.CustNum = cNum WITH FRAME DEFAULT-FRAME:
      oCount = oCount + 1.
      GET NEXT q1.
   END.

   GET PREV q1.
   DISPLAY Customer.CustNum
         Customer.Name
         oCount LABEL "No Of Orders".
   oCount = 0.
   GET NEXT q1 NO-LOCK.

END.

CLOSE QUERY q1.
```



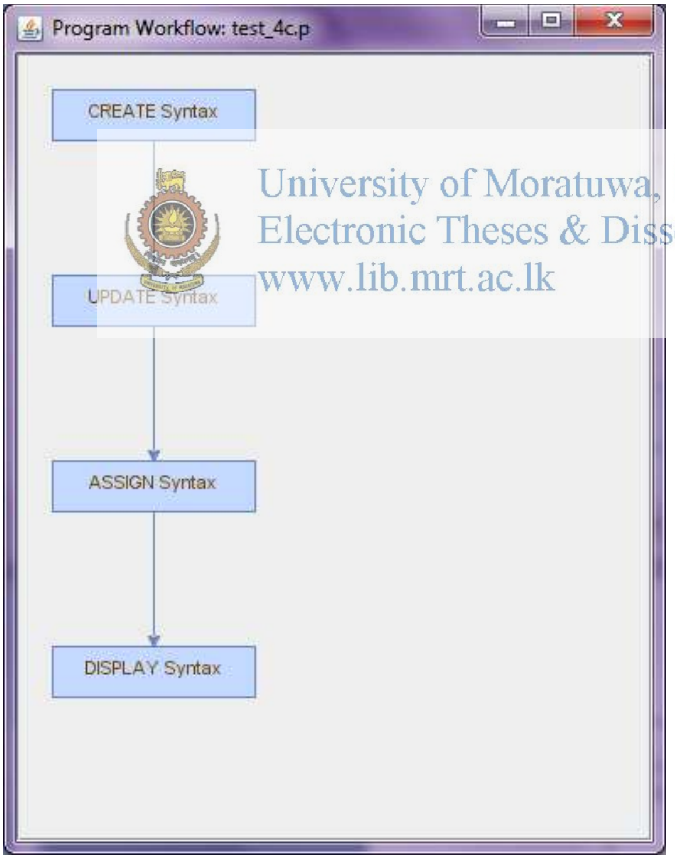
- **Sample Program 7**

```
/*------------------------------------------------------------------------
   File        : test_4b
   Description : Write a procedure which displays the code and name of all customers who have invalid orders, along with the order code and error status id
   Author(s)   : ISubasinghe
   Created     : Wed Mar 17 14:42:48 IST 2010
   Notes       :
 ----------------------------------------------------------------------*/

/* ***************************  Definitions  ************************** */
DEFINE VARIABLE oState     AS CHARACTER.
DEFINE VARIABLE oShipCount AS INT.
DEFINE VARIABLE oOrdCount  AS INT.
DEFINE VARIABLE oBOCount   AS INT.
DEFINE VARIABLE oPSCount   AS INT.
DEFINE VARIABLE oINVCount  AS INT.

/* ***************************  Main Block  *************************** */
FOR EACH Customer, EACH Order OF Customer:
      CASE Order.OrderStatus:
         WHEN "Shipped"          THEN oShipCount = oShipCount + 1.
         WHEN "Ordered"          THEN oOrdCount  = oOrdCount  + 1.
         WHEN "Back Ordered"     THEN oBOCount   = oBOCount   + 1.
         WHEN "Partially Shipped" THEN oPSCount  = oPSCount   + 1.
         OTHERWISE   DO:
                oINVCount = oINVCount + 1.
                DISPLAY Customer.CustNum SKIP
                Customer.Name   SKIP
                Order.Ordernum  SKIP
                Order.OrderStatus.
             END.
      END CASE.
END.

IF oINVCount = 0 THEN
   MESSAGE "No Invalid Orders!" VIEW-AS ALERT-BOX.
```



- **Sample Program 8**

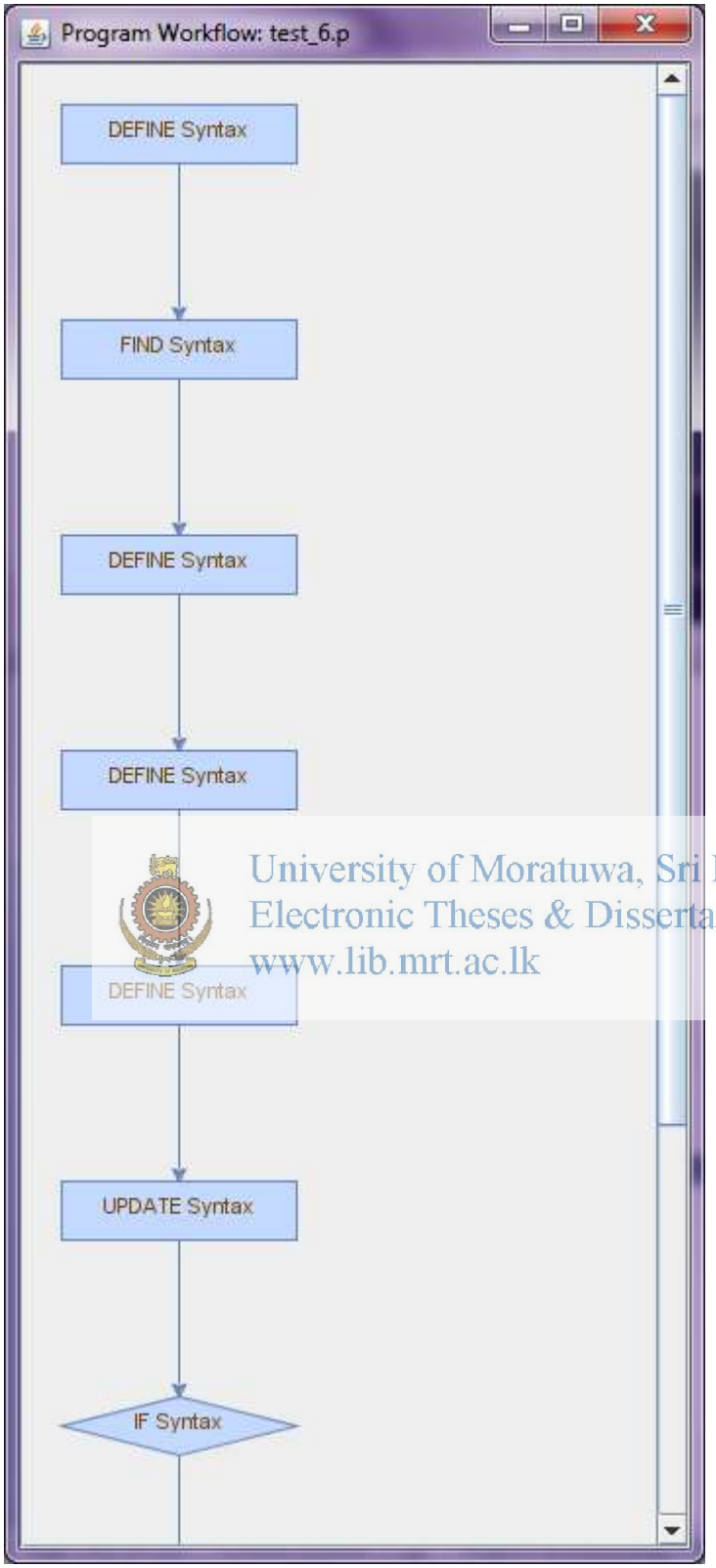
```
/*-----------------------------------------------------------------------
   File       : test_4c.p
   Description : Write a trigger to execute when an order is created, which sets the orderDate to the current date and the deliveryDate to two weeks from the current date
   Author(s)   : ISubasinghe
   Created     : Wed Mar 17 15:16:35 IST 2010
   Notes       :
 -----------------------------------------------------------------------*/

/* ***************************  Main Block  *************************** */
CREATE Order.
UPDATE Order.CustNum.
ASSIGN Order.CustNum.
DISPLAY Order.CustNum
        Order.Ordernum
        Order.OrderDate
        Order.PromiseDate.
```

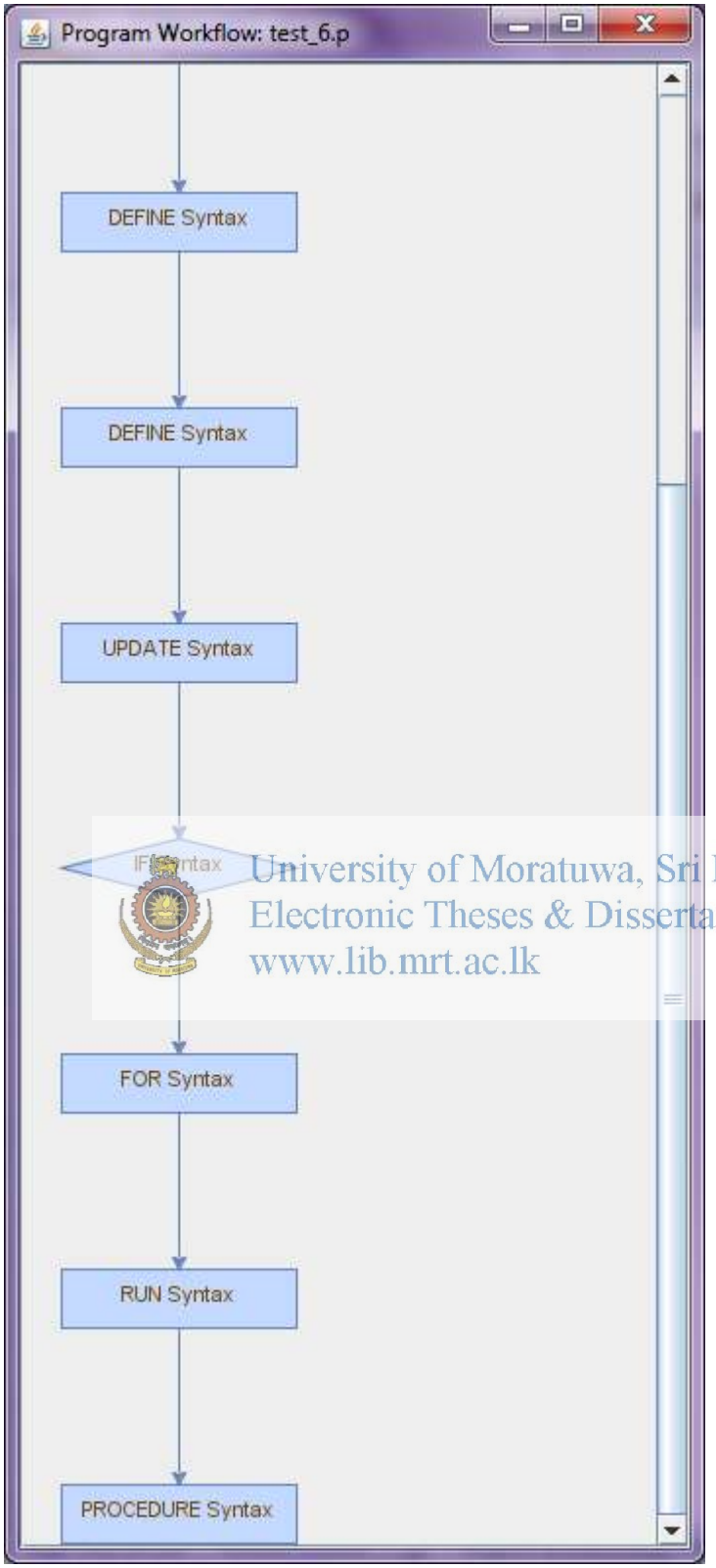


- **Sample Program 9**

```
/*------------------------------------------------------------------------
   File        : test_6.p
   Description :   a. Create temp-tables for the customer, order and orderline tables.
                   b. Write code to accept user input for the customer code, and populate the temp-tables
with related records
                   c. Write the save procedure for the customer temp-table, where the new values are written
back to the
                     database only when the record in the database remains unchanged.
   Author(s)   : ISubasinghe
   Created     : Wed Mar 17 16:55:07 IST 2010
   Notes       :
 ----------------------------------------------------------------------*/

/* ***************************  Definitions  ************************** */
DEFINE TEMP-TABLE ttCustomer  LIKE Customer.
DEFINE TEMP-TABLE ttOrder     LIKE Order.
DEFINE TEMP-TABLE ttOrderLine LIKE OrderLine.

DEFINE VARIABLE cCustNum LIKE Customer.CustNum VIEW-AS FILL-IN.

/* ***************************  Main Block  *************************** */
CURRENT-WINDOW:WIDTH-CHARS = 110.
UPDATE cCustNum LABEL "Customer Number".

FIND FIRST Customer WHERE Customer.CustNum = cCustNum NO-ERROR.

IF AVAILABLE(Customer) THEN
DO:
   CREATE ttCustomer.
   BUFFER-COPY Customer TO ttCustomer.

   FOR EACH Order OF Customer WHERE Order.CustNum = cCustNum NO-LOCK:
      CREATE ttOrder.
      BUFFER-COPY Order TO ttOrder.

      FOR EACH OrderLine OF Order NO-LOCK:
         CREATE ttOrderLine.
         BUFFER-COPY OrderLine TO ttOrderLine.

      END.
   END.
   DISPLAY ttCustomer.CustNum ttOrder.Ordernum.
END.
ELSE
DO:
   CREATE ttCustomer.
   UPDATE ttCustomer.Name
          ttCustomer.CreditLimit
          ttCustomer.Country.
   ASSIGN ttCustomer.CustNum = NEXT-VALUE(NextCustNum)
          ttCustomer.Name
          ttCustomer.CreditLimit
          ttCustomer.Country.
END.
```

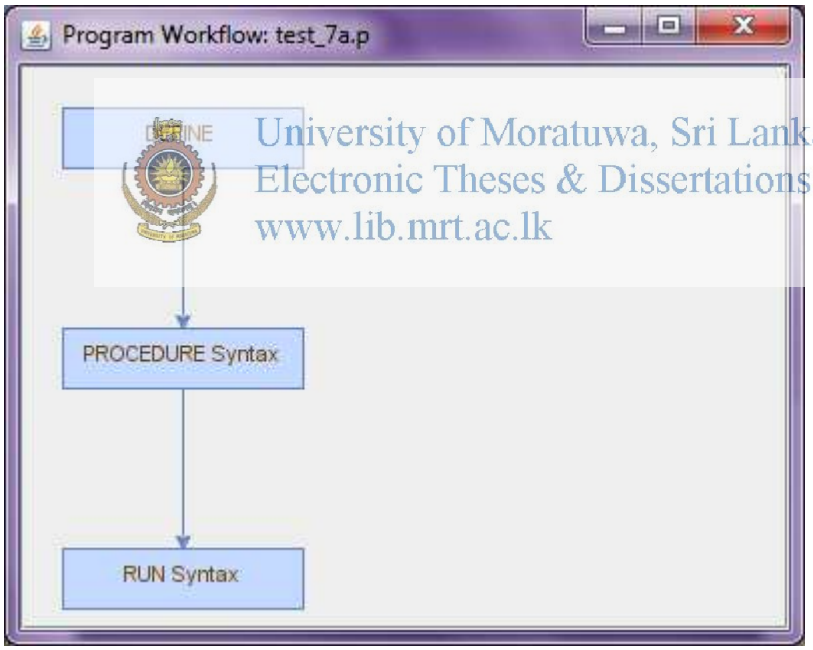
```
FOR EACH ttCustomer NO-LOCK:
   DISPLAY TTcustomer WITH FRAME f1 WIDTH 100.
END.
RUN saveTtCustomer.


PROCEDURE saveTtCustomer:

   FOR EACH ttCustomer:
      FIND Customer WHERE Customer.CustNum = ttCustomer.CustNum EXCLUSIVE-LOCK NO-
ERROR.
         IF CAN-FIND (Customer) THEN

            IF CURRENT-CHANGED (Customer) THEN
               MESSAGE "Customer record has been changed! Cannot Continue to copy."
                     VIEW-AS ALERT-BOX ERROR.
        ELSE
          CREATE Customer.

        BUFFER-COPY ttCustomer TO Customer.
        MESSAGE "Customer " cCustNum " saved!" VIEW-AS ALERT-BOX BUTTON OK.

   END.
END.
```





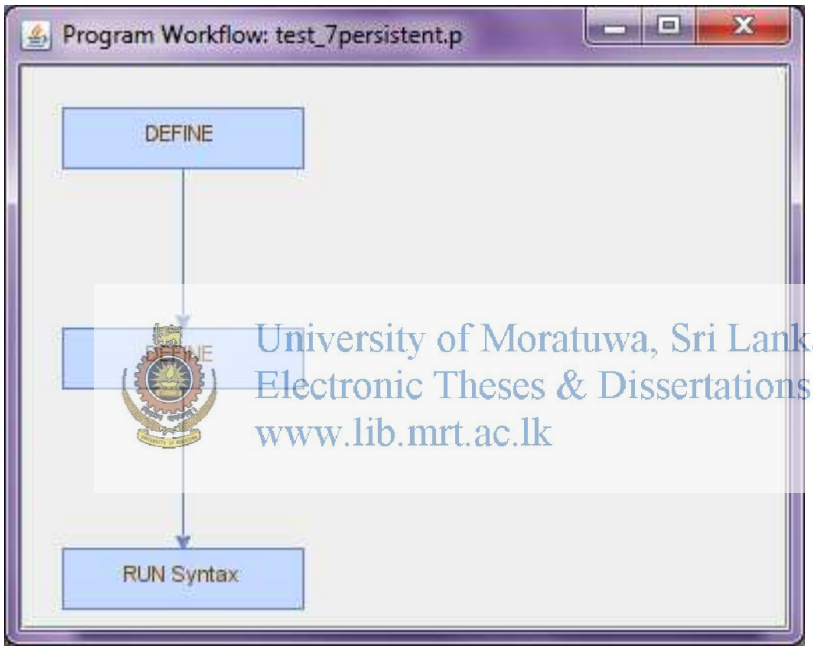
- **Sample Program 10**

```
/*------------------------------------------------------------------------
    File        : test_7a.p
    Description : Write the code for a procedure which calls a persistent procedure, which in turn calls an internal procedure in the parent to display an error message which is sent in by the persistent procedure.
    Author(s)   : ISubasinghe
    Created     : Wed Mar 17 18:11:29 IST 2010
    Notes       :
  ----------------------------------------------------------------------*/

DEFINE VARIABLE hParent AS HANDLE.

/* ***************************  Main Block  *************************** */
RUN test_7persistent.p PERSISTENT (INPUT THIS-PROCEDURE).


PROCEDURE displayError:
    DEFINE INPUT PARAMETER msg AS CHARACTER.

    MESSAGE msg VIEW-AS ALERT-BOX ERROR.
END.
```



- **Sample Program 11**

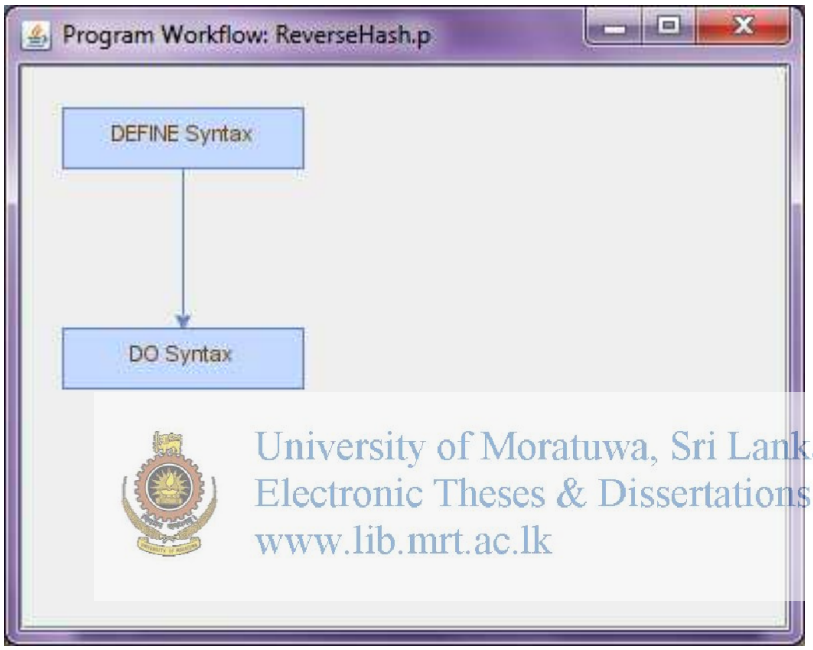
```
/*------------------------------------------------------------------------
    File        : test_7persistent.p
    Author(s)   : ISubasinghe
    Created     : Wed Mar 17 18:22:17 IST 2010
    Notes       :
  ----------------------------------------------------------------------*/

/* ***************************  Definitions  ************************** */
DEFINE VARIABLE errMsg AS CHARACTER INITIAL "My Message from test_7persitent.p!".
DEFINE INPUT PARAMETER hParent AS HANDLE.

RUN displayError IN hParent (INPUT errMsg).
```



- **Sample Program 12**

```
/* ------------------------------------------------------------------ */
/* Program name : ReverseHash.p                                       */
/* Description  : Decrypt hash to return word generated through hash () */
/*              where base value is 7 and multiplier wis 47.          */
/* Syntax       : RUN ReverseHasp.p(INPUT int64HashNum).              */
/* Author       : Ilakshini Subasinghe                                */
/* Date created : 06-MAR-2015                                         */
/* ------------------------------------------------------------------ */

/* ************** I/O Param ************** */
DEFINE INPUT  PARAMETER ipi64HashValue  AS INT64 NO-UNDO.

/* ************** Forward Declarations ************** */
FUNCTION fnGetChar RETURNS CHAR (INPUT-OUTPUT iReverseHash AS INT64)
FORWARD.

/* ************** Internal Variables ************** */
DEFINE VARIABLE  i            AS INTEGER                  NO-UNDO.
DEFINE VARIABLE  iReverseHash AS INT64 INITIAL 371580748701652777 NO-UNDO.
DEFINE VARIABLE  word         AS CHARACTER                NO-UNDO.
DEFINE VARIABLE  iBase        AS INTEGER   INITIAL "7"    NO-UNDO.

/* ************** MAIN ************** */

ASSIGN iReverseHash = ipi64HashValue. /* Backup Value */

/* Get each letter from hash (Returns last letter first) */
DO WHILE (iReverseHash > iBase):
   ASSIGN word = fnGetChar(INPUT-OUTPUT iReverseHash) + word.
END.

/* Display Result */
MESSAGE "Your Word Embedded in" ipi64HashValue "is:" SKIP word
   VIEW-AS ALERT-BOX.

/* ************** END MAIN ************** */


/* ************** Function Declarations ************** */
FUNCTION fnGetChar RETURNS CHARACTER (INPUT-OUTPUT iReverseHash AS INT64):
/* ------------------------------------------------------------------ */
/* Function  : fnGetChar                                              */
/* Purpose   : Returns Character Hidden in Hash Number.               */
/* Parameters: i/p: Hash Number                                       */
/* ------------------------------------------------------------------ */
   DEFINE VARIABLE i             AS INTEGER              NO-UNDO.
   DEFINE VARIABLE i64Remainder  AS INT64                NO-UNDO.
   DEFINE VARIABLE cLetters      AS CHARACTER            NO-UNDO.
   DEFINE VARIABLE iBase         AS INTEGER   INITIAL "7"   NO-UNDO.
   DEFINE VARIABLE iMultiplier   AS INTEGER   INITIAL "47"  NO-UNDO.

   ASSIGN
      cLetters = "abcdeghilmnoprstuwy".
```

```
   IF iReverseHash = iBase THEN
      RETURN "".

   DO  i = 1 TO LENGTH(cLetters):
      i64Remainder = iReverseHash - i.
      IF i64Remainder MODULO iMultiplier   = 0 THEN
      DO:
         ASSIGN
            iReverseHash = i64Remainder / iMultiplier.
         RETURN SUBSTRING(cLetters,i,1).
      END.
   END.
END.
```



APPENDIX B: CLASSIFICATION OUTPUT

Rules.NNge (WEKA)

```
=== Run information ===

Scheme:weka.classifiers.rules.NNge -G 5 -I 5
Relation:     4GLkeywords
Instances:    34
Attributes:   4
              token_type_parent
              token_type_child1
              token_type_child2
              mxgraph_vertex_shape
Test mode:evaluate on training data

=== Classifier model (full training set) ===

NNGE classifier

Rules generated :
        class loop IF : token_type_parent in {DO,FOR,PRESELECT,PROMPT-FOR,UPDATE} ^
token_type_child1 in {PRESELECT,TO,EACH,EDITING} ^ token_type_child2 in {FIRST,?}  (6)
        class loop IF : token_type_parent in {DO,FOR,PRESELECT} ^ token_type_child1 in
{FOR,FIRST,LAST,PRESELECT} ^ token_type_child2 in {EACH,LAST}  (8)
        class rhombus IF : token_type_parent in {DISPLAY,DO,PROMPT-FOR,UPDATE,IF} ^
token_type_child1 in {WHEN,QUERY-TUNING,WHILE,CAN-DO} ^ token_type_child2 in {?}  (6)
        class rhombus IF : token_type_parent in {ASSIGN,CASE} ^ token_type_child1 in {WHEN,?}
^ token_type_child2 in {?}  (2)
        class default IF : token_type_parent in
{DISPLAY,DEFINE,DO,FOR,PRESELECT,PROMPT-FOR,UPDATE} ^ token_type_child1 in
{VARIABLE,TEMP-TABLE,BUFFER,FOR,FIRST,LAST,?} ^ token_type_child2 in {FIRST,?}
(12)

Stat :
        class default : 1 exemplar(s) including 1 Hyperrectangle(s) and 0 Single(s).
        class rhombus : 2 exemplar(s) including 2 Hyperrectangle(s) and 0 Single(s).
        class loop : 2 exemplar(s) including 2 Hyperrectangle(s) and 0 Single(s).

        Total : 5 exemplars(s) including 5 Hyperrectangle(s) and 0 Single(s).

        Feature weights : [0.5212479644597324 1.1260791090720865 0.41986782197619227]



Time taken to build model: 0 seconds

=== Predictions on training set ===

inst#,    actual, predicted, error, probability distribution
    1  1:default  2:rhombus     +   0     *1      0
    2  1:default  1:default        *1      0      0
    3  1:default  1:default        *1      0      0
    4  1:default  1:default        *1      0      0
    5  1:default     3:loop     +   0      0     *1
```

```
    6  1:default  1:default       *1     0     0
    7  1:default    3:loop     +  0     0    *1
    8  1:default    3:loop     +  0     0    *1
    9  1:default    3:loop     +  0     0    *1
   10  1:default    3:loop     +  0     0    *1
   11  1:default    3:loop     +  0     0    *1
   12  1:default    3:loop     +  0     0    *1
   13  2:rhombus  2:rhombus        0    *1     0
   14  2:rhombus  2:rhombus        0    *1     0
   15  2:rhombus  2:rhombus        0    *1     0
   16  2:rhombus  2:rhombus        0    *1     0
   17  2:rhombus  2:rhombus        0    *1     0
   18  2:rhombus  2:rhombus        0    *1     0
   19  2:rhombus  2:rhombus        0    *1     0
   20  2:rhombus  2:rhombus        0    *1     0
   21    3:loop     3:loop         0     0    *1
   22    3:loop     3:loop         0     0    *1
   23    3:loop     3:loop         0     0    *1
   24    3:loop     3:loop         0     0    *1
   25    3:loop     3:loop         0     0    *1
   26    3:loop     3:loop         0     0    *1
   27    3:loop     3:loop         0     0    *1
   28    3:loop     3:loop         0     0    *1
   29    3:loop     3:loop         0     0    *1
   30    3:loop     3:loop         0     0    *1
   31    3:loop     3:loop         0     0    *1
   32    3:loop     3:loop         0     0    *1
   33    3:loop     3:loop         0     0    *1
   34    3:loop     3:loop         0     0    *1

=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances          26               76.4706 %
Incorrectly Classified Instances         8               23.5294 %
Kappa statistic                          0.6334
Mean absolute error                      0.1569
Root mean squared error                  0.3961
Relative absolute error                 36.0976 %
Root relative squared error             85.0462 %
Total Number of Instances               34

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.333    0          1          0.333    0.5        0.667    default
                 1        0.038      0.889      1        0.941      0.981    rhombus
                 1        0.35       0.667      1        0.8        0.825    loop
Weighted Avg.    0.765    0.153      0.837      0.765    0.727      0.806

=== Confusion Matrix ===

  a  b  c   <-- classified as
  4  1  7 |  a = default
  0  8  0 |  b = rhombus
  0  0 14 |  c = loop
```

Bayes.NaiveBayes (WEKA)

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     4GLkeywords
Instances:    34
Attributes:   4
              token_type_parent
              token_type_child1
              token_type_child2
              mxgraph_vertex_shape
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

Naive Bayes Classifier

                Class
Attribute         default rhombus    loop
                (0.35)  (0.24)  (0.41)
==========================================
token_type_parent
  DISPLAY             2.0     2.0     1.0
  DEFINE              4.0     1.0     1.0
  DO                  3.0     3.0     7.0
  FOR                 3.0     1.0     4.0
  PRESELECT           3.0     1.0     4.0
  PROMPT-FOR          2.0     2.0     2.0
  UPDATE              2.0     2.0     2.0
  ASSIGN              1.0     2.0     1.0
  CASE                1.0     2.0     1.0
  IF                  1.0     2.0     1.0
  [total]            22.0    18.0    24.0

token_type_child1
  VARIABLE            2.0     1.0     1.0
  TEMP-TABLE          2.0     1.0     1.0
  BUFFER              2.0     1.0     1.0
  FOR                 2.0     1.0     3.0
  FIRST               3.0     1.0     3.0
  LAST                3.0     1.0     3.0
  WHEN                1.0     5.0     1.0
  QUERY-TUNING        1.0     2.0     1.0
  WHILE               1.0     2.0     1.0
  CAN-DO              1.0     2.0     1.0
  PRESELECT           1.0     1.0     4.0
  TO                  1.0     1.0     2.0
  EACH                1.0     1.0     3.0
  EDITING             1.0     1.0     3.0
  [total]            22.0    21.0    28.0

token_type_child2
  FIRST               2.0     1.0     2.0
  EACH                1.0     1.0     7.0
  LAST                1.0     1.0     3.0
  [total]             4.0     3.0    12.0
```

Time taken to build model: 0 seconds

=== Predictions on training set ===

inst#	actual	predicted	error	prediction
1	1:default	1:default		0.421
2	1:default	1:default		0.823
3	1:default	1:default		0.823
4	1:default	1:default		0.823
5	1:default	3:loop	+	0.572
6	1:default	1:default		0.441
7	1:default	3:loop	+	0.502
8	1:default	3:loop	+	0.502
9	1:default	3:loop	+	0.502
10	1:default	3:loop	+	0.502
11	1:default	3:loop	+	0.364
12	1:default	3:loop	+	0.364
13	2:rhombus	2:rhombus		0.829
14	2:rhombus	2:rhombus		0.451
15	2:rhombus	2:rhombus		0.758
16	2:rhombus	3:loop	+	0.412
17	2:rhombus	3:loop	+	0.412
18	2:rhombus	2:rhombus		0.659
19	2:rhombus	2:rhombus		0.708
20	2:rhombus	2:rhombus		0.708
21	3:loop	3:loop		0.81
22	3:loop	3:loop		0.892
23	3:loop	3:loop		0.619
24	3:loop	3:loop		0.646
25	3:loop	3:loop		0.78
26	3:loop	3:loop		0.673
27	3:loop	3:loop		0.72
28	3:loop	3:loop		0.696
29	3:loop	3:loop		0.696
30	3:loop	3:loop		0.72
31	3:loop	3:loop		0.696
32	3:loop	3:loop		0.696
33	3:loop	3:loop		0.569
34	3:loop	3:loop		0.569

=== Evaluation on training set ===

Time taken to test model on training data: 0.08 seconds

=== Summary ===

Correctly Classified Instances	25	73.5294 %
Incorrectly Classified Instances	9	26.4706 %
Kappa statistic	0.5785	
Mean absolute error	0.2605	
Root mean squared error	0.314	
Relative absolute error	59.9568 %	
Root relative squared error	67.4255 %	
Coverage of cases (0.95 level)	100 %	
Mean rel. region size (0.95 level)	91.1765 %	
Total Number of Instances	34	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.417	0.000	1.000	0.417	0.588	0.562	0.977	0.972	default
0.750	0.000	1.000	0.750	0.857	0.835	1.000	1.000	rhombus

```
               1.000    0.450    0.609      1.000    0.757      0.579    0.993     0.990     loop
Weighted Avg.  0.735    0.185    0.839      0.735    0.721      0.633    0.989     0.986

=== Confusion Matrix ===

  a  b  c   <-- classified as
  5  0  7 |  a = default
  0  6  2 |  b = rhombus
  0  0 14 |  c = loop
```

Trees.J48 (WEKA)

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     4GLkeywords
Instances:    34
Attributes:   4
              token_type_parent
              token_type_child1
              token_type_child2
              mxgraph_vertex_shape
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

J48 pruned tree
------------------

token_type_child1 = VARIABLE: default (1.17/0.03)
token_type_child1 = TEMP-TABLE: default (1.17/0.03)
token_type_child1 = BUFFER: default (1.17/0.03)
token_type_child1 = FOR: loop (3.52/1.52)
token_type_child1 = FIRST: default (4.69/2.14)
token_type_child1 = LAST: default (4.69/2.14)
token_type_child1 = WHEN: rhombus (4.69/0.55)
token_type_child1 = QUERY-TUNING: rhombus (1.17/0.14)
token_type_child1 = WHILE: rhombus (1.17/0.14)
token_type_child1 = CAN-DO: rhombus (1.17/0.14)
token_type_child1 = PRESELECT: loop (3.52/0.52)
token_type_child1 = TO: loop (1.17/0.17)
token_type_child1 = EACH: loop (2.34/0.34)
token_type_child1 = EDITING: loop (2.34/0.34)

Number of Leaves  :     14

Size of the tree :      15


Time taken to build model: 0.01 seconds

=== Predictions on training set ===

 inst#     actual  predicted error prediction
    1  1:default    3:loop   +   0.412
    2  1:default  1:default       0.971
    3  1:default  1:default       0.971
    4  1:default  1:default       0.971
    5  1:default    3:loop   +   0.412
    6  1:default    3:loop   +   0.569
```

```
    7  1:default  1:default      0.544
    8  1:default  1:default      0.544
    9  1:default  1:default      0.544
   10  1:default  1:default      0.544
   11  1:default    3:loop   +   0.412
   12  1:default    3:loop   +   0.412
   13  2:rhombus  2:rhombus      0.882
   14  2:rhombus    3:loop   +   0.412
   15  2:rhombus  2:rhombus      0.882
   16  2:rhombus  2:rhombus      0.882
   17  2:rhombus  2:rhombus      0.882
   18  2:rhombus  2:rhombus      0.882
   19  2:rhombus  2:rhombus      0.882
   20  2:rhombus  2:rhombus      0.882
   21    3:loop     3:loop      0.569
   22    3:loop     3:loop      0.853
   23    3:loop     3:loop      0.853
   24    3:loop     3:loop      0.569
   25    3:loop     3:loop      0.853
   26    3:loop     3:loop      0.853
   27    3:loop     3:loop      0.853
   28    3:loop  1:default   +   0.544
   29    3:loop  1:default   +   0.544
   30    3:loop     3:loop      0.853
   31    3:loop  1:default   +   0.544
   32    3:loop  1:default   +   0.544
   33    3:loop     3:loop      0.853
   34    3:loop     3:loop      0.853

=== Evaluation on training set ===

Time taken to test model on training data: 0.07 seconds

=== Summary ===

Correctly Classified Instances          24               70.5882 %
Incorrectly Classified Instances        10               29.4118 %
Kappa statistic                          0.543
Mean absolute error                      0.2161
Root mean squared error                  0.3034
Relative absolute error                 49.7226 %
Root relative squared error             65.1435 %
Coverage of cases (0.95 level)         100      %
Mean rel. region size (0.95 level)      68.6275 %
Total Number of Instances               34

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.583    0.182    0.636      0.583   0.609      0.410    0.852     0.720     default
               0.875    0.000    1.000      0.875   0.933      0.918    0.990     0.958     rhombus
               0.714    0.300    0.625      0.714   0.667      0.408    0.954     0.912     loop
Weighted Avg.  0.706    0.188    0.717      0.706   0.709      0.529    0.926     0.855

=== Confusion Matrix ===

  a  b  c   <-- classified as
  7  0  5 |  a = default
  0  7  1 |  b = rhombus
  4  0 10 |  c = loop
```

KNearestNeighbour (Java-ML)