An Intelligent tool for converting Prolog problems into Prolog codes

Intelligent Prolog Pseudocode Converter (IPPC)

By

Wang Keng Kuen

Department of Artificial Intelligent
Faculty of Computer Science and Information Technology
University Malaya.

Submitted in partial fulfillment of the
requirements for the Degree of
Bachelor of Computer Science
2002/2003

Abstract

A thesis presented in partial fulfillment of the requirements for the Bachelor of Computer Science degree. This document is submitted Dr. Rukaini Haji Abdullah (thesis supervisor) and Ms Norisma Idris (thesis moderator), lecturers of FSKTM as a report for the Final Year Project Level One, WXES3181.

Intelligent Prolog Pseudocode Converter (IPPC) are designed to allow novice programmer have an easier tool for them to get Prolog codes converted from the Pseudocode they had wrote. IPPC included documents to help guide users and teaching them about how to write program with using IPPC.

The first part of the thesis presents about problems definitions for using Prolog in writing program and introduction of IPPC. The second part introduces the theoretical basis of the research and the methodologies employed to apply them. The final part described about the IPPC's requirement analysis and IPPC's design.

For the next advancement stage, WXES3182, the Final Year Project Level One, WXES3181 is important as references in the actual coding and development in WXES3182. However, there may have be some changes to the system design in this proposal for WXES3182. The changes have been reflecting in the product and in this report.

Acknowledgment

I would like to acknowledge the help of many people during my preparation of this thesis. At first, I would like to express my utmost gratitude to Dr. Rukaini Haji Abdullah (thesis supervisor), for helping to supervise me, providing advices and subjects, and offering direction and penetrating criticism. In addition, I would like to thank the subject, WXES3181, provided opportunity to get experiences that I regard as so important.

Secondly, I would like to thank my project moderator, Ms Norisma Idris for being so informative during the VIVA session. The advices, questions and feedback were so useful for this thesis. Best wishes for her recovery from illness.

I have also benefited from many discussions with my friends and course mates where I had a chance to develop some of my ideas before this thesis commenced.

Finally yet importantly, I would like to give my thanks and loves to my family.

Table of Contents

List of Figures

List of Tables

Chapter One

Introduction

There are some controversial views that historically accompanied Prolog. Prolog fast gained popularity in Europe as a practical programming tool but still not familiar in Malaysia. In Japan, Prolog was placed at the central of the development of the fifth-generation computer.

For conventional language are procedural oriented, Prolog introduces the descriptive or declarative, view. This greatly alters the way of thinking about problems and makes learning to student of computer science should learn something about Prolog at some point because Prolog enforces a different problem solving paradigm complementary to other programming languages [1].

Prolog is known to be a difficult language to master. It does not have the familiar control primitives used by languages like RATFOR, ALGOR and PASCAL so the system does not give too much help to the programmer to employ structured programming concepts. In addition, many programmers have become used to strongly typed languages. Prolog is very weakly typed indeed. This gives the programmer great power to experiment but carries the obvious responsibility to be careful [2].

For preparation in coding phase, programmer will write down paper works like Pseudocode (in natural language). From the Pseudocode, programmer must translate it into correct syntax. However, most of the novice programmer will confuse with what of the predicates and arguments should be (for a fact). Beside that, novice programmer also will make mistake in define rule (head :- body). They will confuse with what suppose be the head (conclusion) or body (premises).

Alternatively, there are three main difficulties in writing programs: (1) the difficulty specifying the problem from its natural language description, (2) the difficulty in transforming a given specification into an algorithm or procedure, and (3) the difficulty in writing the algorithm, in a target language [5].

Thereby, is good to have a tool called "Intelligent Prolog Pseudocode Converter (IPPC)" for helping programmers solving their problems in writing Prolog programs.

1.1 What is an “Intelligent tool for convert Pseudocode into Prolog codes” / IPPC?

Most of the programming language has a lot of syntax that the user must obey before a program is build. Normally a programmer will use some technique to make their works easier, like they will write down a Pseudocode or flowchart before get into coding phase. For this reason, why not just build a tool which can translate the Pseudocode into codes for wasting time on thinking about how to write it in coding phase. For accomplish this notion, a tool name “Intelligent Prolog Pseudocode Converter / IPPC” is good to be develop.

IPPC is a tool that can translate Pseudocode (in natural language form) into prolog codes. This tool is built by using Visual Basic 6 as System Development Software and connected with a database which contains useable keywords. Refer figure 1.2 for more detail about IPPC.

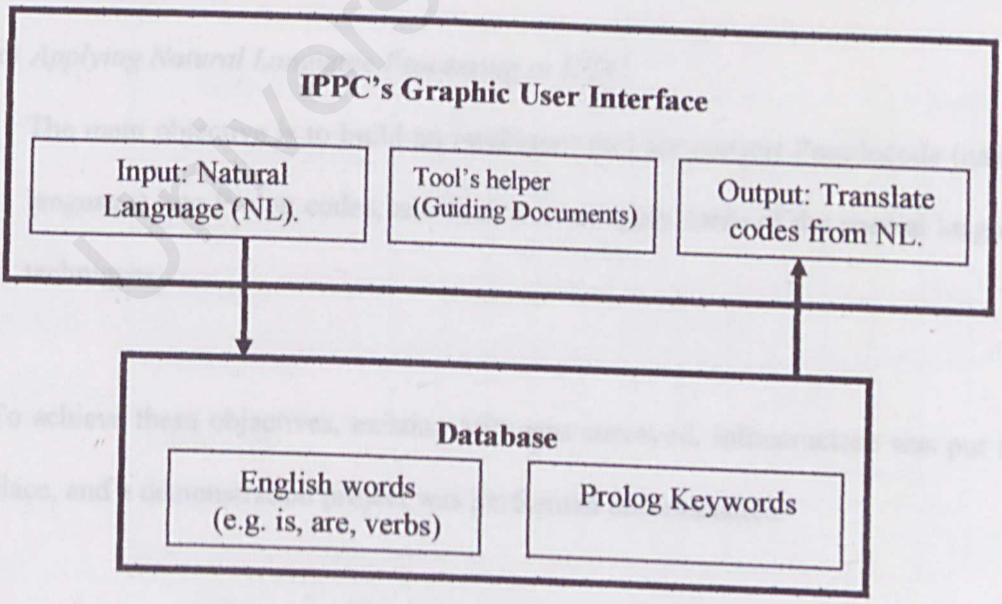

IPPC's Graphic User Interface
Input: Natural Language (NL).
Tool's helper (Guiding Documents)
Output: Translate codes from NL.
Database
English words (e.g. is, are, verbs)
Prolog Keywords

Figure 1.1 Main component of IPPC.

1.2 Project Objectives

There are some reasons or objectives for proposed this tool.

a) Helping novice programmer from confusion and learn more about programming language in Prolog.

IPPC can use as codes checker. Novice programmer can check whether they have defined codes in correct syntax. Beside that IPPC can popularize usage of Prolog because it just need user to input Pseudocode (natural language).

b) Convert Pseudocode into an executable Prolog programs.

For avoid time recklessly in coding phase, programmer should write down their idea within a document or paper work in Pseudocode style. From the paper work, they just need to key in all the Pseudocode using IPPC. Then it will automatically convert the Pseudocode into executable Prolog codes.

c) Applying Natural Language Processing in IPPC.

The main objective is to build an intelligent tool for convert Pseudocode (natural language) into Prolog codes, and then it must apply some of the natural language techniques.

To achieve these objectives, existing VP5 was surveyed, infrastructure was put into place, and a demonstration project was performed and evaluated.

1.3 Scope and limitation

In this thesis, the main purpose is to build a tool which can convert Prolog Pseudocode into Prolog codes. Moreover, the tool is only compatible for who are wanted to build program through Visual Prolog version 5.2 (VP5). IPPC is a converter and not a compiler, thereby, it need cooperate with VP5 for run the converted codes (from IPPC).

By the way the user must have understanding in English because the Pseudocode is written in English. In addition, the user must have understanding in grammar, which is a formal specification of the structures allowable in the language [6]. IPPC is limited with words set in the database, which converting progress will success if the words (in Pseudocode) are match with the words in database.

1.4 Timeline

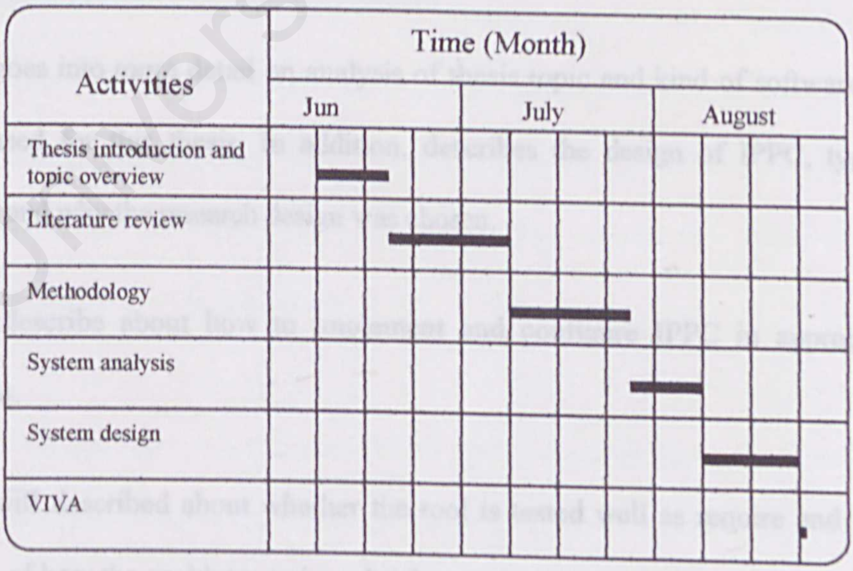


Figure 1.2 Gantt chart for Chapter 1 to Chapter 4.

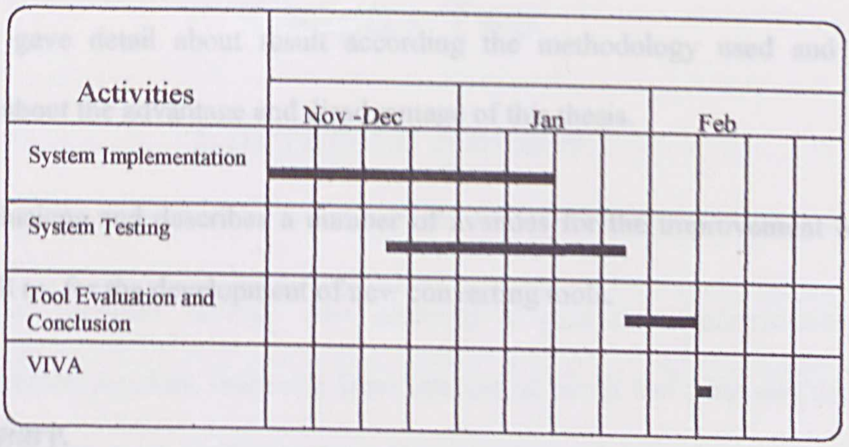


Figure 1.3 Gantt chart for Chapter 5 to Chapter 7.

1.5 Thesis structure.

The project research will be encountered in the following chapters.

Chapter 2 is literatures review which about research existed proposed literature same as this thesis. In addition, brief information related to this topic.

Chapter 3 is about explanation of technique and processes used for this thesis.

Chapter 4 goes into more detail on analysis of thesis topic and kind of software and hardware used for this thesis. In addition, describes the design of IPPC, type of design used and why the research design was chosen.

Chapter 5 describe about how to implement and configure IPPC in appropriate environment.

Chapter 6 will described about whether the tool is tested well as require and some explanation of how the problems to be solved.

Chapter 7 gave detail about result according the methodology used and some discussion about the advantage and disadvantage of this thesis.

Final conclusions and describes a number of avenues for the improvement of this tool, as well as, for the development of new converting tools.

1.6 Summary.

This chapter has described some problems met by user in using Prolog. In this thesis, it gives a brief statement at the outset of the objectives of this thesis and the main conclusions. Beside that, scope and limitation were described and schedule of doing this thesis was illustrate on the timeline.

This chapter is very useful to readers before they plunge into more detailed descriptions. In the next chapter, more explanations of particular point how to make this thesis succeed will be presented to readers.

Chapter Two

Literature Review

Research and literature reviews were methods to gain more information from proposed articles, journals, reference from internet or books and proposed thesis in FSKTM document room. With that some discussion related will be describes in this chapter.

This chapter will cover some topics like some research, what is intelligent, introduction of Natural Language, characteristics of VP5, appropriate programming languages and DBMS.

2.1 Some research.

From definitions of a dictionary, convert are means (1) To change or turn from one state or condition to another; to alter in form, substance, or quality; to transform; to transmute; as, to convert water into ice. (2) To turn into another language; to translate and so on. For this thesis, convert is mean change the Pseudocode in Natural language form to Prolog codes.

There are too many types of converter, which some is already in used and some are still in research. For examples, C-language sources to HTML converter, MS-WORD to HTML converter, Convert files to pseudo-natural-language text and back again

[13], and so on. However, most of these converters are not converting natural language to code, like C-language sources to HTML converter or c2html is a syntax highlighter for C source code that produces a highlighter html file as output. For MS-WORD to HTML converter or MSWordView is a program that can understand the Microsoft word 8 binary file format (office97), it currently converts word into html, which can then be read with a browser, thereby, it also not corresponding to this thesis.

In addition, a converter called NICETEXT is a package that converts ciphertext or any input file into pseudo-natural-language text and recovers the ciphertext or file from the text. The expandable set of tools allows experimentation with custom dictionaries, automatic simulation of writing style, and the use of Context-Free-Grammars to control text generation. It is the result of Masters Thesis research at the University of Wisconsin, Milwaukee under the advisement of Dr. George Davida [14].

2.2 What is Intelligence?

Intelligence is easier to recognize than to define or measure. While the word "intelligence" is used in ordinary conversation, and has a dictionary definition, it has no agreed-upon scientific meaning, and no quantitative natural law relating to intelligence have as been discovered. A dictionary definition of intelligence includes statements such as (1) ability to meet (novel) situation successfully by proper behavior adjustment; or (2) the ability to perceive the interrelationships of presented facts in such a way as to guide action toward a desired goal [7].

For this thesis, intelligence in IPPC is mean to make a tool that can analyze the sentences (object, predicates or relation, and facts) and convert it into the Prolog codes without Prolog syntax errors occur.

2.3 Pseudocode Definition.

A series of statements that outline what a computer program will do, without putting in the actual programming code. The Pseudocode, which can be written in natural language, is a preliminary step in designing a program, and helps the programmer think through what steps will be necessary. At the Pseudocode stage, it is not necessary to know what programming language will be used. In a later step, the Pseudocode will be translated into actual computer [9].

```
IF Total > 10 THEN
    SET Carry to 1
    SUBTRACT 10 from Total
ELSE
    SET Carry to 0
END IF
```

Figure 2.1 Pseudocode example

2.4 Natural language understanding.

Prolog is an appropriate programming language for natural language, but not used in full because programmer still need to understand some syntax for express the meaning of the sentences. Pseudocode is a paper work written in partial or fully natural language. In addition, IPPC apply Pseudocode as main inputs from user. Therefore is good to understand some knowledge about sentences structure and grammars.

2.4.1 Different levels of language analysis

In IPPC, user must use considerable knowledge about the structure of the language, including what the words are, how words combine to form sentences, what the words mean, how words meanings contribute to sentence meanings.

In natural language understanding, it have some of the different forms of knowledge, e.g. phonetic and phonological knowledge, morphological knowledge, syntactic knowledge, semantic knowledge, pragmatic knowledge, discourse knowledge and world knowledge.

For understanding in this thesis, syntactic and semantic knowledge were used. Synthetic knowledge concern how words can be put together to form a correct sentences and determines what structural role each words play in the sentences and what phrases are subpart of other phrases. For semantic knowledge, it concern what words mean and how these meanings combine in sentences to form sentences meanings [6].

For build an IPPC, the tool must have ability to understanding the meaning of the sentences given by user. Thereby IPPC must involve with algorithm of natural language syntax and semantic.

2.4.2 Classes of words

The basic unit in a sentence is derived from words, which can group to 4 main classes – nouns (N), adjectives (ADJ), verbs (V), and adverbs (ADV). Each of this class has their own means. Beside that, some others classes like articles, pronouns, preposition, particles, quantifiers, conjunctions, and so on also will give different means of sentence given.

For noun phrases (NPs) was used to refer to things: objects, places, concepts, events, quality, and so on (most of these NPs define as facts in clause section). The simplest NP consists of a single pronoun: he, she, they, you, me, it, I and so on. Another form of noun phase consists of a name or proper noun, such as Wang or Mike. These nouns will initial in capitalized form.

A sentence consists of an NP, the subject and followed by verb phase (VP), the predicate. A simple VP consists of some adverbial modifiers followed by the head verb and its complements. Verbs have different classes: like auxiliary verbs (be, do and have), modal verbs (will, can and could) and main verbs (eat, ran and believe). From these three classes, main verbs normally will be treating as predicate in facts or rules.

Therefore, it is important that an IPPC can recognize the predicate or arguments should be, by identifying the classes of natural language.

2.4.3 Grammars and sentence structure

For examine syntactic structure of a sentence, two things must be consider the grammar, which is a formal specification of the structures allowable in the language and parsing technique, which is the method pf analyzing a sentence determine its structure according to the grammar [6].

To form a sentence, the sentence must consist of an initial noun phase (NP) and followed verb phase (VP). From the VP, it composes with a V and an NP (which consist of article and common noun) [3]. For example, John ate the cat was a sentence group by NP – V – Art – N. On the other hand, this sentence can represented using tree representation as below:

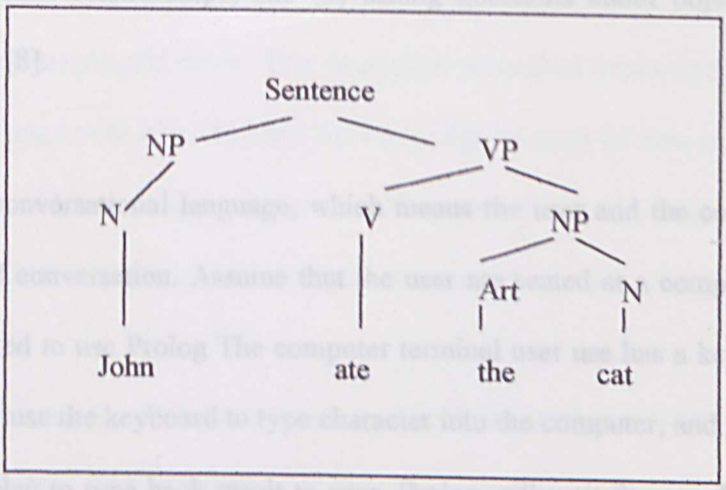


Figure 2.2 Sentence basic structures.

From the structure, IPPC will recognize which will be the facts, predicates, and arguments to form the correct codes in Visual Prolog.

2.5 Programming Logic.

Programming Logic or Prolog invented by Alain Colmerauer and Phillipe Roussel at the University of Aix-Marseille in 1971, and addressed to avoid problems of combinatorial explosion: it can be viewed as a restricted theorem prover, where the restrictions are on the input language, on the inference rules used and on the search strategy. Prolog is a descriptive or declarative language, which means a series of step specifying how the computer must work to solve a problem, a Prolog program consists of a description of the problem [4]. Alternatively, Prolog is a computer programming language that is for solving problems that involve objects and relationships between objects. Computer programming in Prolog consist of (1) declaring some facts about objects and their relations, (2) defining some rules about objects and their relationships, and (3) asking questions about objects and their relationships [8].

Prolog is a conversational language, which means the user and the computer carry out a kind of conversation. Assume that the user are seated at a computer terminal and have asked to use Prolog The computer terminal user use has a keyboard and a display. User use the keyboard to type character into the computer, and the computer uses the display to type back result to user. Prolog will wait for user to type in the facts and rules that pertain to the problem user want to solve. Then, if user asks the right kind of question, Prolog will work out the answers and show them on the display.

Advantages using Prolog: (1) Prolog can make deduction from the existed facts or rules. (2)Prolog execution is controlled automation. Prolog is a powerful and has a very short and simple syntax which can reduce human errors and maintenance cost. (3) Scoping rules are simple and uniform in Prolog and declaration of variable names is not required. This reduces code size and opportunities for error. (4) Prolog has a history of use for linguistics research and natural language processing. (5) Prolog is not a complete implementation of logic; it is much closer to it than other programming languages like C.

Disadvantages using Prolog: (1) It tempts user to write things that look logically correct, but that won't run. (2) The obvious way to write a predicate is unlikely to be efficient. Users must know when a predicate needs to be optimized. (3) Because it lacks functional notation, predicates can become cumbersome. (4) Input and output is not always easy. (5) There are some features which have not been standardized, and differ between implementations. For examples, formatted input and output, file-handling, sorting predicates. (6) User can't re-assign to parts of data structures. This makes it impossible to implement arrays. However, functional programmers have developed a number of fast-access data structures which do almost as good a job.

Early implementations included C-Prolog, ESLPDPRO, Frolic, LM-Prolog, Open Prolog, SB-Prolog, UPMAIL Tricia Prolog. In 1998, the most common Prologs in use are Quintus Prolog, SICSTUS Prolog, LPA Prolog, SWI Prolog, AMZI Prolog, SNI Prolog and Visual Prolog. Because of VP5 (personal edition) is a freeware system available in http://www.visual-prolog.com and have a friendly programming environment, therefore is good to use it as thesis resource in IPPC development.

2.5.1 Fundamentals of Prolog.

Below will introduce some important element of Prolog, which is facts, rules, variables and goals or questions.

2.5.1.1 Facts.

Most of the facts are objects. For example, "John likes Mary", which this sentence consist of two objects, called "John" and "Mary".and a relationship, called "likes". In Prolog, this facts will be write in standard form like this: likes(john, mary).

There have some important things to be mention about is: (1) the names of all relationships and objects must begin with a lower-case letter. For examples, likes, john, mary. (2) The relationships is written first, and then objects are written separated by commas, and the objects are enclosed by a pair of round brackets. (3) The full stop character '.' must come at the end of a fact.

Beside that, for defining relationship between objects using facts, user should pay attention to what order the objects are written between the round brackets. However, the order is arbitrary and must solve consistently. For current arbitrary convention, the "liker / John" from the above fact will be put as the first of the two objects in round brackets and the object that is liked (Mary) in second slot. Therefore, the fact likes (john, mary). which mean "John likes Mary" is not same thing as likes (mary, john). with meaning "Mary likes John".

For some terminology, the names of the objects that are enclosed within the round brackets in each fact are called arguments (John or Mary). Moreover, name of the relationship, which comes just before the round brackets, is called predicate (likes).

In prolog, a collection of facts is called a database. Therefore, the word database shall be use whenever collected facts are used to solve particular problem.

2.5.1.2 Rules.

Sometime in sentence like "John likes Mary" is easy to define as fact in Prolog, then how about a sentence "John likes all people"? One way to do this would be to write down separate facts like likes (john, Alfred), likes (john, charles), and so on for every person in the database. However, this becomes tedious, especially if there are hundreds of people in the Prolog program.

Another way to solve this problem is using rules. In Prolog, rules are used when a fact depends on a group of other facts. On the other hand, rules are also use to express definitions. A rule is a general statement about objects (some use variables for objects) and relationships For example: John likes anyone who likes wine or in other words John likes X if X likes wine.

In Prolog, a rule is consist of a head (conclusion part: left-hand side of the rule) and a body (condition part: right-hand side of the rule). The head and body are connected by the symbol :-, pronounced as *if.* For examples as above sentence, is written in Prolog as: likes (john, X) :- likes (X, wine). Notice that the rules are ended with a dot.

For solving a question in the rules, matching (predicate are the same for a fact) to the database will occur.

2.5.1.3 Variables.

Sometime the user may want to find out what things that John likes, so the user may ask "Does John like books?", "John likes Mary?", and so forth, with Prolog giving a Yes-or-No answer each time. It is tiresome. So is more sensible to ask Prolog to tell the user something that John likes. Then user can phase a question of this form as "Does John likes X?". Because the users do not know what the object is that X stand for, so Prolog will give what the possibilities are. In Prolog, users can use names like X, which called as variable, to stand for objects to be determined by Prolog. For example, likes (john, X).

When Prolog uses a variable, the variables can either instantiated or not instantiated. A variable is instantiated when there is an object that the variable stands for. A variable is not instantiated when what the variable stands for is not yet known. Prolog can distinguish variables from names of particular objects because any name beginning with a capital letter is taken to be variable.

Beside this, Prolog also has an anonymous variable which used " _ " or underscore as a symbol. This variable will filter the information not need to be show on screen (refer figure 2.3).

Variables in Prolog have some distinctions from other programming languages, which variables are not store information and the value it gets is through pattern-matching to constants (arguments) in facts or rules. A variable said to be free before it gets a value; when it get a value, it become bound. Only bound until a solution is obtained then prolog will unbind it, backup and look for alternatives solutions.

```
PREDICATES
......
CLAUSES
   male(bill).
   male(joe).
   female(sue).
   female(tammy).
   parent(bill,joe).
   parent(sue,joe).
   parent(joe,tammy).

GOAL
   parent(Parent, _).

   Parent=bill
   Parent=sue
   Parent=joe
```

Figure 2.3 Example of anonymous variable.

2.5.1.4 Queries.

After all of the facts and rules are defined in the program, users can ask the program for get something or result from that program. For this, Prolog has one part call – Queries or Goal for user input their questions to get results.

When a question likes: likes (john, mary) is asked of Prolog, it will search through the database (facts). It looks for facts that match the fact in the question. The fact match if their predicate is the same and their corresponding arguments each are same. If Prolog finds a fact that matches the question, Prolog will respond yes. If no such facts exist in the database, Prolog responds no.

If a variable used in the question like likes (john, X), Prolog will search the database and instantiate the variable with some value or object. If the predicate and argument was matched, Prolog will respond by giving the instantiated value on the variable likes X = mary.

2.5.2 Prolog's Syntax.

Prolog provides way to structure data as well as to structure the order in which attempts are made to satisfy goals. Structuring data involves knowing the syntax by which users can denote data.

Syntax of a language describes how users are allowed to fits words together. In English, the syntax of the sentence "I see a zebra" is correct but the syntax of "zebra see I a" is not correct.

For Prolog, the programs are built from terms. A term is a constant, variable, or a structure. Characters are divided into 4 categories which are (1) upper-case letter, (2) lower-case letter, (3) digits, and (4) sign characters.

Constants are specific objects or specific relationships. Have two kind of constant: atom and integers. There are two kind of atom: those make up of letters and digits (must begin with lower-case letter), and those make up from sign (enclosed in single quotes ‘ ’ ’ with any characters in it or insert underline character ‘_’ within the atom). For examples, void, ‘str3ing’, george_smith and so on. The other kind of constant: Integers, which used to represent numbers for arithmetic operation.

The second kind of term used in Prolog is the variable. Variables look like atom, except it begins with capital letter or an underline sign.

The third of term with which Prolog programs are written is the structure. A structure is a single object which consists of a collection of other objects, call components. The components are group together into a single structure for convenience in handling them. For example, an index card for a library book contain several components like the author's name, the title of the book, the date when it was published, location to find it and so forth. A structure is written in Prolog by specifying its functor, and its components. The components is enclosed in round brackets and separated by commas. The functor is written just before the opening round bracket. For example, owns (john, book (wuthering_heights, bronte), which owns fact have a structure by the name of book and two component, a title and an author.

2.5.3 Visual Prolog's program sections.

Generally, a Visual Prolog program includes few basic program sections. These are the clauses section, the predicates section, the domains section, the goal section and another few common section likes facts, constant and global.

Clauses section is the heart of a Visual Prolog program which facts and rules are defined in it. In satisfy a goal, Visual Prolog will proceed at clauses section to search appropriate match. As Visual Prolog proceeds down through the clauses section, it places internal pointers next to each clause that matches the current sub-goal. If that clause is not part of a logical path that leads to a solution, Visual Prolog returns to the set pointer and looks for another match (Backtracking).

The predicates section is part for declaration of predicates and the domains (types) of the arguments to the predicates. This is important for telling the Visual Prolog what things inside the program. For predicate declaration, it must begin with predicate name followed by its arguments (in a parenthesis). Each argument type separate by comma and predicate declaration is not followed by a period (same like below).

```
predicateName(argument_type1, argument_type2, ..., argument_typeN)
```

After define rules and facts in clauses section and declared predicates in predicates section, user can start to make a query in goal section. The query is same as a rule but not followed by :- and it simply a list of sub-goals. This section will be automatically executed by Visual Prolog when the program runs. If all arguments a query is succeed then program will terminate successful, vice-versa program is said to have fail.

For a big program write in Visual Prolog with a large number of predicates, some of the argument types are making the codes more implicit or unclear. Because of this, in Visual Prolog have a domain section that allow user to declare corresponding argument types using meaningful words and limit the argument in some domain only. From example, a sentence like "Frank is a male who is 45 years old" which will give a declaration predicate as below:

```
PREDICATES

person (symbol, symbol, integer)
```

Can be change to

```
DOMAINS

          name, sex = symbol

          age       = integer

PREDICATES

          person (name, sex, age)
```

Therefore, Visual Prolog will easily detect error occur if the argument in "person" swap which argument types will not match.

Same as others programming languages, Visual Prolog also provide a constant section for declaring a fix numeral value to a meaningful word (syntax: <Id> = <Macro definition>). Then user can save time in rewrite the same complex calculation or numerals and make the program more easily to read. Declaration in constant section is like below:

```
CONSTANTS

        hundred = (10*(10-1)+10)

        pi   = 3.141592653
```

Most of the facts are define in clauses section before get in runtime, but how about adding new facts when runtime occur? For this problem, Visual Prolog has a facts section (keyword facts is synonymous with database) which is a part of dynamic database that allow user update (change, remove, or add) some of the program facts. Standard predicates or keywords like assert, asserta, assertz and consult is used to adding new facts at runtime. Retract and retractall used for delete facts from the program at runtime.

Most of the section describe above is a local definition and declaration for a project, then how about to make a project with modules in it? So for this a section calls global section (rather than local), which is used for enable communication across different modules. For large project is good to write each section in a separate module for make easily in enhancements or modifications. In this section, user can write global domain, global predicates and global facts for use in a module by include it (refer figure 2.2).

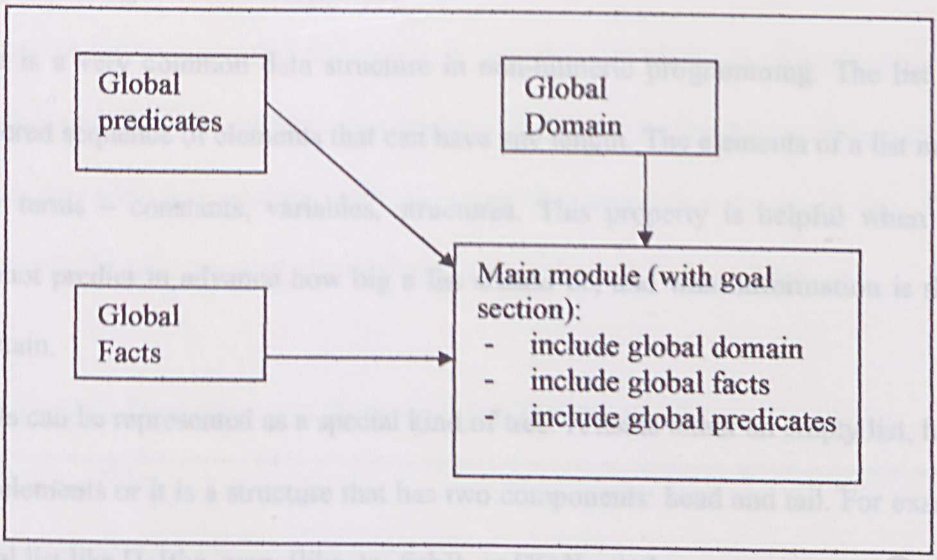


Figure 2.4 Illustrate communications between module (main) with sub-module (global predicates, global domains, and global facts).

2.5.4 Built-in Predicates in Visual Prolog.

Within a Visual Prolog, it has a lot of predefined keywords that easily to use by user. Visual Prolog has categorized the built-in predicates into few parts: Arithmetic Functions and Predicates, Control Predicates, Conversions, Data Compression, Error & Break Control, External Database System, File System, Binary Handling, Input/Output, Internal Facts Sections, Machine Low-level, Miscellaneous, OS Related, and String Handling. For detail about built-in predicates in Visual Prolog can visit to http://www.visual-prolog.com.

2.5.5 Prolog's data structures.

List is a very common data structure in non-numeric programming. The list is an ordered sequence of elements that can have any length. The elements of a list may be any terms – constants, variables, structures. This property is helpful when users cannot predict in advance how big a list should be, and what information is should contain.

Lists can be represented as a special kind of tree. A list is either an empty list, having no elements or it is a structure that has two components: head and tail. For example, legal list like [], [the, men, [like, to, fish]], or [X+Y, x+y].

Common operation with list is split a list into its head and tail. For example, the list with head X and Tail Y will be written as [X|Y], where the symbol '|' separate the head and tail. List also use for representing strings. For example, a string "system" will changed by Prolog into a list [115, 121, 115, 116, 101, 109].

From a list, some operation like (1) membership, which is checking whether come object is an element of a list, which corresponds to checking for the set membership, (2) concatenation of two list, obtaining a third list, which may correspond to the union of sets, (3) adding a new object to a list, or deleting some object form it.

2.5.6 Cut.

A 'cut' allow users to tell Prolog which previous choices it need not consider again when it backtrack though the chain of satisfied goals. Syntactically, 'cut' is representing as '!' (called predicate and no arguments) in Prolog program.

There have three main area of using 'cut', which is (1) telling the Prolog system that it has found the right rule for a particular goal, (2) telling the Prolog system to fail a particular goal immediately without trying for alternative solutions, and (3) to terminate the generation of alternative solutions through backtracking.

2.6 System Development Software for IPPC.

There are so many of system development software for develop new system for use in industrial, banking, military or research. Most popular system development software used by Computer Company is C and C++, COBOL, Java, UML, RPG, Visual Prolog, and Visual Basic.

For this project, Visual Basic 6 (Visual Basic descendent) was chose for develop the IPPC tool. Visual Basic provides a complete set of tools to simplify rapid application development. So what is Visual Basic? The "Visual" part refers to the method used to create the graphical user interface (GUI). Rather than writing numerous lines of code to describe the appearance and location of interface elements, users simply add pre-built objects into place on screen. If users have ever used a drawing program such as Paint, users already have most of the skills necessary to create an effective user interface. Addition, Visual Basic is an object-based programming language.

The "Basic" part refers to the BASIC (Beginners All-Purpose Symbolic Instruction Code) language, a language used by more programmers than any other language in the history of computing. Visual Basic has evolved from the original BASIC language and now contains several hundred statements, functions, and keywords, many of which relate directly to the Windows GUI. Beginners can create useful applications by learning just a few of the keywords, yet the power of the language allows professionals to accomplish anything that can be accomplished using any other Windows programming language [10].

Visual Basic has the tools like (1) Data access features allow users to create databases, front-end applications, and scalable server-side components for most popular database formats, including Microsoft SQL Server and other enterprise-level databases. (2) ActiveX™ technologies allow users to use the functionality provided by other applications, such as Microsoft Word processor, Microsoft Excel spreadsheet, and other Windows applications. Users can even automate applications and objects created using the Professional or Enterprise editions of Visual Basic. (3) Internet capabilities make it easy to provide access to documents and applications across the Internet or intranet from within your application, or to create Internet server applications, and (4) finished application is a true .exe file that uses a Visual Basic Virtual Machine that users can freely distribute.

2.7 Database.

A database is a collection of information organized as to make it easy to view it, search it, retrieve the right detail, and collects the necessary facts in an easier, timely, and effortless manner as possible [15].

Databases fall into two broad categories: (1) desktop and scalable applications such as Microsoft Access, and (2) true RDBMS (Relational Database Management Systems) such as Oracle, Informix, and SQL Server.

Microsoft Access is a relational database used on desktop computers to manage information on different levels for different purposes. Microsoft Access can be used for personal information management, in a small business to organize and manage all data, or in an enterprise to communicate with servers.

There have some advantages use of Microsoft Access, which are no extra server cost, ease of implementation, lower development cost and compatibility with existing desktop system. Also have few disadvantages like limited amount data storage, not a long term option for a large site.

For this thesis, database is used for store some keywords correspond to the tool usage like sign symbols (:- , '.', and so forth). Thereby, it not need too big of storage and the tool is use for local PC only.

Chapter Three

Methodology

More information, better information, and better management of information are part of software development request. Because the requests and the need far exceed the resources limited productivity and limited number of hours in the day. That led to a discussion of methodology. Methodology means the science of method or arrangement; a treatise on method. Therefore, this chapter will draw out the suitable methodology for the proposed tool.

Beside that, this chapter will describe approach, model, tool, problem, and domain involved in the methodology.

3.1 Process Model.

A well-designed tool will follow one of the proscribed software process models. A process model is also known as a software-engineering paradigm. It is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required.

There are so many process models for software development. For examples, V-model, Transformation model, Prototyping model, Waterfall model and so on. For this thesis, waterfall model with prototyping had chosen for guiding the development process.

The model for development process was proceeding from identifying problems to implementation and maintenance, as shown in figure 3.1.

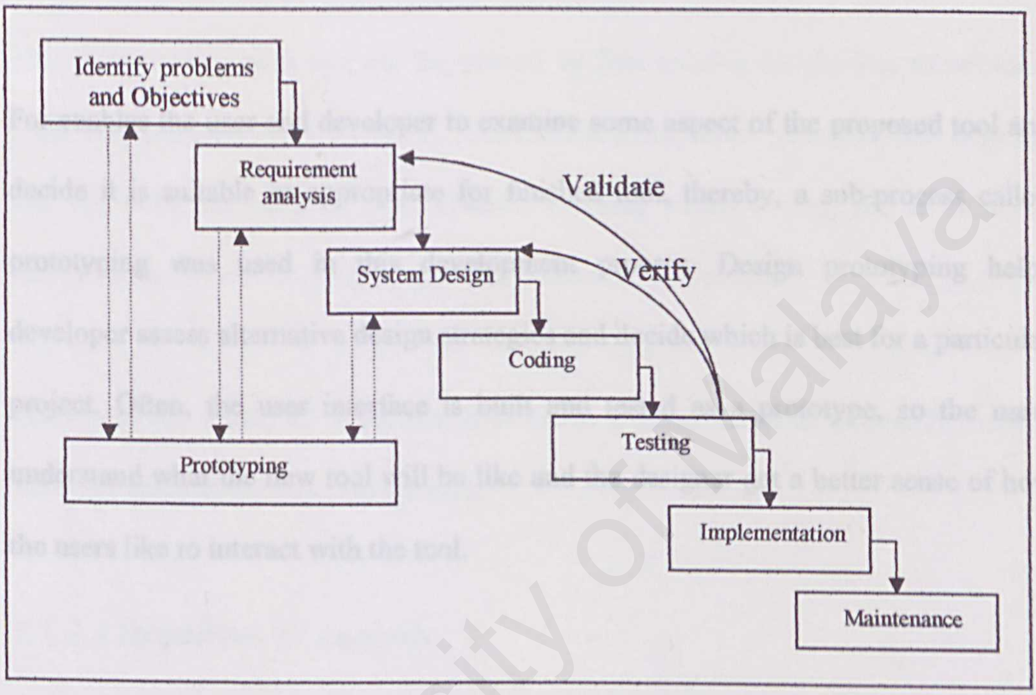


Figure 3.1 Waterfall Model With Prototyping.

3.1.1 Discussion of Waterfall model With Prototyping.

The Waterfall model abstracts the essential activities of software development and lists them in their most primitive sequence of dependency. Real development projects (software and other) rarely follow such a model literally, mainly because the model can and is applied to itself recursively, yielding an almost fractal fabric of actual activity.

For enables the user and developer to examine some aspect of the proposed tool and decide it is suitable or appropriate for finished tool, thereby, a sub-process called prototyping was used in this development process. Design prototyping helps developer assess alternative design strategies and decide which is best for a particular project. Often, the user interface is built and tested as a prototype, so the users understand what the new tool will be like and the designer get a better sense of how the users like to interact with the tool.

3.1.2 Activities for development process.

In the Waterfall model with prototyping, have few activities for development process which is identifying problems, requirement analysis, system design, coding, testing, implement, operation and maintenance, and a sub-process, prototyping.

3.1.2.1 Identifying problems.

This thesis begins with scoping the project by first gaining insight into definitions, problems, and objectives for IPPC. From this activity, some questions like why the tool was built, the usage of tool, difficulty of using Visual Prolog, and so on about problems met by novice programmer had identify in this phase.

Thereby, after identified problems, it come to expand the problems with feasible solutions and step for develop the tool.

3.1.2.2 Requirement analysis.

Before a tool start build, it begins by establishing requirements for all tool elements and then allocating some subset of these requirements to tool. The requirements gathering process is intensified and focused specially on tool. To understand the nature of the tool to be built, the system engineer ("analyst") must understand the information domain for the tool, as well as required function, behavior, performance and interfacing. The essential purpose of this phase is to find the need and to define the problem that needs to be solved. It also includes the personnel assignments, costs, project schedule, and target dates.

3.1.2.3 System Design.

In this phase, the tool's overall structure and its nuances are defined. The databases design, the data structure design, and the number of tiers needed for the package architecture etc are all defined in this phase. Analysis and design are very crucial in the whole development cycle. Any glitch in the design phase could be very expensive to solve in the later stage of the tool development. Much care is taken in this phase.

3.1.2.4 Coding.

The design must translate into a machine readable form. The coding phase performs this task. If design is performed in detail manner, code generation can be accomplished with out much complication. Programming tool like compilers, interpreters, debuggers are used to generate the codes. For this thesis, VB6 was used as software development system to generate the code for the tool.

3.1.2.5 Testing.

Once the code is generated, the program testing begins. Different testing methodologies are available to unravel the bugs that were committed during the previous phases. Different testing tools and methodologies are already available.

3.1.2.6 Implementation.

After the testing is completed, the tool will send to user. User will use the for real situation and

3.1.2.7 Maintenance.

The tool will definitely undergo change once it is delivered to the user. There are many reasons for the change. Change could happen because of unexpected input values into the system. In addition, the changes in the system could directly affect the tool operations. The tool should be developed to accommodate changes that could happen during the post implementation period.

3.1.3 Advantages and Disadvantages of Waterfall model With Prototyping.

	Waterfall model	Prototyping model
Advantages	• Good way to sketch a plan for the development. It will likely be too simple, and usually a good, durable framework for the real plan. • The stages are clear cut. • All R&D done before coding starts implies better quality program design. • Well suited for developing stable, well understood computer-based applications.	• User sees a real system fast (1 or 2 days). • User directly involved in specifying requirements. • Made for change therefore possibly low maintenance cost. • Serves as a basis for discussion and helps identify requirements when there is no current system like desired system.
Disadvantages	• The waterfall process is possible for experienced developers if application domain is very well understood by the developer, developer has a detailed knowledge of and practiced skills in using the production tools and finally the developer has experience of the successful development of similar products before. • Client must wait until the end to see any product. • Can be too rigid. • One phase must be completed before proceeding on to the next.	• Requires high upfront costs (software for database, modeling, report, generation, screen generation.) • Difficult to use when building large systems. • Sometimes difficult to maintain user enthusiasm. • User never satisfied. • Tendency not to document.

Table 3.1 advantages and disadvantages for waterfall model and prototyping model.

3.2 Prototype for this thesis.

For user understanding, is good to build a user interface which will give some notion about the tool would be like. Thereby, below was some of the detail about IPPC (Figure 3.2).

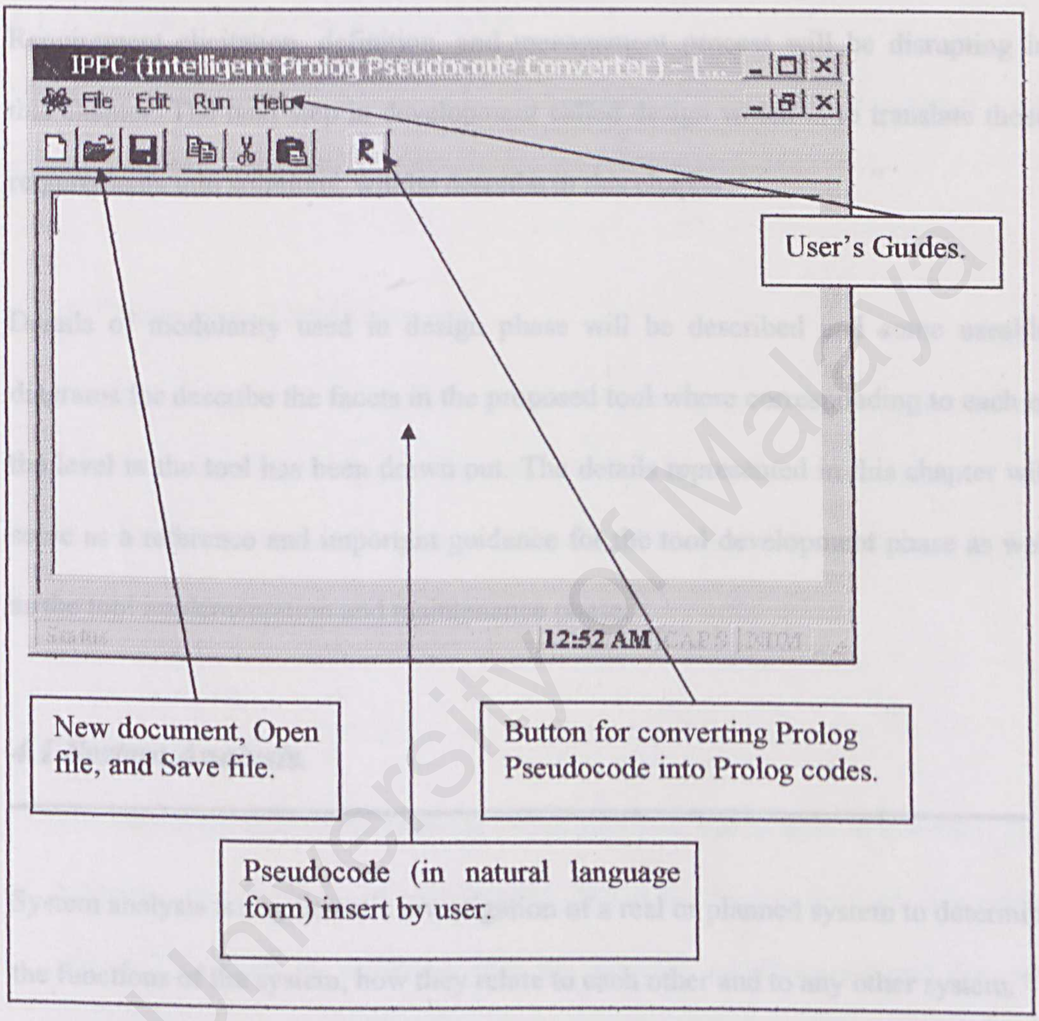

IPPC (Intelligent Prolog Pseudocode Converter) - [...
File
Edit
Run
Help
Status
12:52 AM
CAPS
NUM
User's Guides.
New document, Open file, and Save file.
Button for converting Prolog Pseudocode into Prolog codes.
Pseudocode (in natural language form) insert by user.

Figure 3.2 Interface for IPPC.

Chapter Four

System Analysis and Design

This chapter will discuss about some system analysis and the design of the system. Requirement elicitation, definition, and management process will be disrupting in this chapter. The next step in development called design which is to translate those requirements into solutions, will be describe in this chapter.

Details of modularity used in design phase will be described and some useable diagrams for describe the facets in the proposed tool where corresponding to each of the level in the tool has been drawn out. The details represented in this chapter will serve as a reference and important guidance for the tool development phase as well as the tool implementation and maintenance phase.

4.1 System Analysis.

System analysis is a systematic investigation of a real or planned system to determine the functions of the system, how they relate to each other and to any other system.

Each proposed model of the software development process includes activities aimed at capturing requirement which also occurred in waterfall model with prototyping. Defining the requirement is the first and most critical, step in software system development. If the requirements are done well, the software flows logically and smoothly. Conversely if the requirements are done poorly, the resulting design is awkward and the coding is more difficult. Usually, errors identified in the requirement stage are the fastest and least expensive to correct, while those found in later stages are increasingly more time-consuming and expensive to correct.

4.1.1 System requirements analysis.

There have two ways for describe requirements which are functional and nonfunctional requirements.

4.1.1.1 Functional Requirement.

A functional requirement describes an interaction between the system and its environment and described how the system should behave given certain stimuli [16]. For this thesis, it consists of 2 groups of functional requirement which are (1) User requirements and (2) Tool requirements.

(1) User requirements.

The user requirements are used (in connection with the System in question) as a complete, user-level set of requirements. When the tool is finished, it must be proven by means of validation that the external specifications and the user requirements have been met.

a. User interface.

For allowing user interact with the tool, is good to have a friendly tool's interface. The interface must have:

i. Forms for user insert Pseudocode.

Allow user to key in the Pseudocode, and allow user open few forms in same tool's working space.

ii. Toolbar and menus.

For making the tool ease to use, it must attach with icon and shortcut for user to click on it. For examples, save icon, copy, cut and paste icon.

iii. Tool Guides (Help).

Some important description about how to use the tool and some constraint and knowledge about the tool will be shown in the help section.

iv. Result displayer.

The result converted from the Pseudocode must appear in the Visual Prolog 5.2 working space or display on a form which can save in Visual Prolog's file type.

(2) Tool requirements.

This part will give some specify requirements of the tool:

a. Description of functionality.

i. The tool will receive a list of Pseudocode from user input.

ii. User must click the 'Run' icon for converting Pseudocode into Prolog codes.

iii. The codes generated from the tool will be shown in Visual Prolog 5.2 work space or shown in tool's output form.

b. Data Constraint.

i. A single list of Pseudocode be generate on time.

ii. User can't easily input unrecognized words which are not inside the database. On the other hand, the words will be converting into related predicate, argument, or variables which contribute to execution failure.

iii. The Pseudocode must as short as possible and must end with period '.'

c. Data operations.

i. The user can open many forms for writing Pseudocode, but one form (a list of Pseudocode) will be converting into Prolog codes on time.

ii. The user can save the Pseudocode into specified file type and also the Prolog codes generated into Prolog file type, by selecting buttons shown on the tool.

iii. The user can create new list of Pseudocode, view existed Pseudocode, and edit or delete a Pseudocode from the tool.

iv. The user can find some help from help documents inside the tool.

4.1.1.2 Nonfunctional Requirement.

A non-functional requirement is a description of other features, characteristics and constraints that define a satisfactory system [16]. Below are the non-functional requirements of the tool:

i. Maintainability

Maintainability is the degree to which the tool can be cost-effectively made to perform its functions in a possibly changing operating environment. The tool are easy to modify and test in updating process to meet the new request, correcting errors, or move to a different computer system.

ii. Reliability

The tool operates in a user-acceptable and does not produce dangerous or costly failure when it is applied in a reasonable manner.

iii. Response Time and Performance

The time to convert Pseudocode to Prolog codes must be within a reasonable time. However, the performance especially the question "how fast will it give the answer?" is very much depending on the hardware used. It will be very slow if the power of processor is low and vise versa. Fortunately, recent computer hardware becomes much more efficiency. Therefore, the response time could consider acceptable.

iv. User friendliness

The design of the GUI must able to attract users' focus and easy to use.

4.2 System Design.

Design development involves those processes and methods involved in translating those identified detailed functional task elements as well as broader organizational goals and requirements into concrete interface objects and dynamic interaction techniques. In addition, design is the creative process of transforming the problem into a solution. To design a tool is to determine a set of components and inter-component interfaces that satisfy a specified set of requirements [16].

IPPC design solution follows by four levels. These three levels involve (1) architecture, (2) IPPC converting process, (3) IPPC's Database structures and (4) Graphic User Interface design. Design development may involve developing completely new interface/interaction solutions as well as retro fitting existing systems by providing a more effective interface and interaction.

4.2.1 Level 1 - Architecture.

This level associates the system capabilities identified in the requirements specification with the system components that will implement them. Component usually modules and the architecture also describes the interconnections among them [16].

IPPC's components consist of input from user, intermediate converting process (algorithm, input, and output), and IPPC's database (refer Figure 4.1).

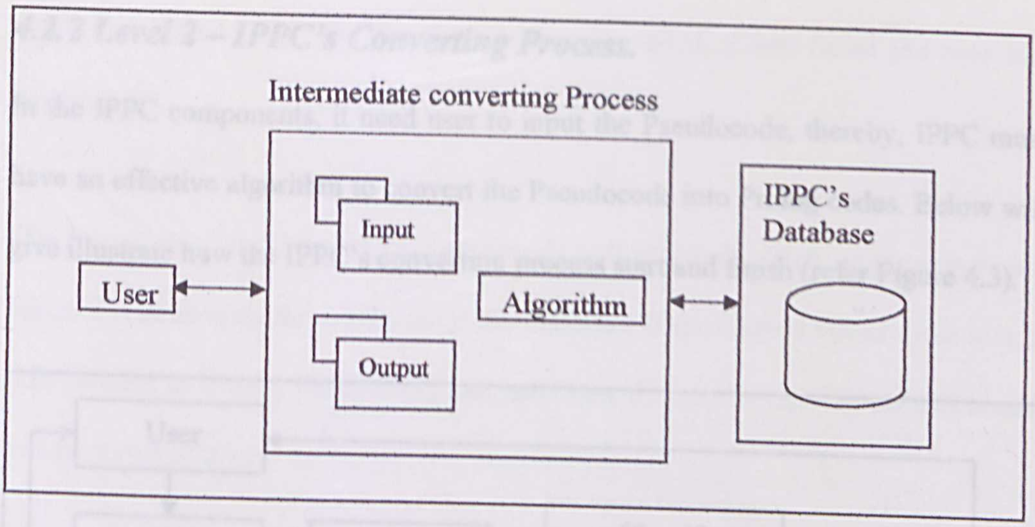


Figure 4.1 IPPC's Physical Architecture.

From the figure shown above, the user needs to input the Pseudocode into IPPC. After all Pseudocode was input, intermediate converting process (algorithm) will start (described in level 2). The process will using matching technique which matches some identified words with words in IPPC's Database. Result from the intermediate converting process will be show back to user with IPPC output screen or Visual Prolog 5.2 workspace (refer Figure 4.2 for more detail).

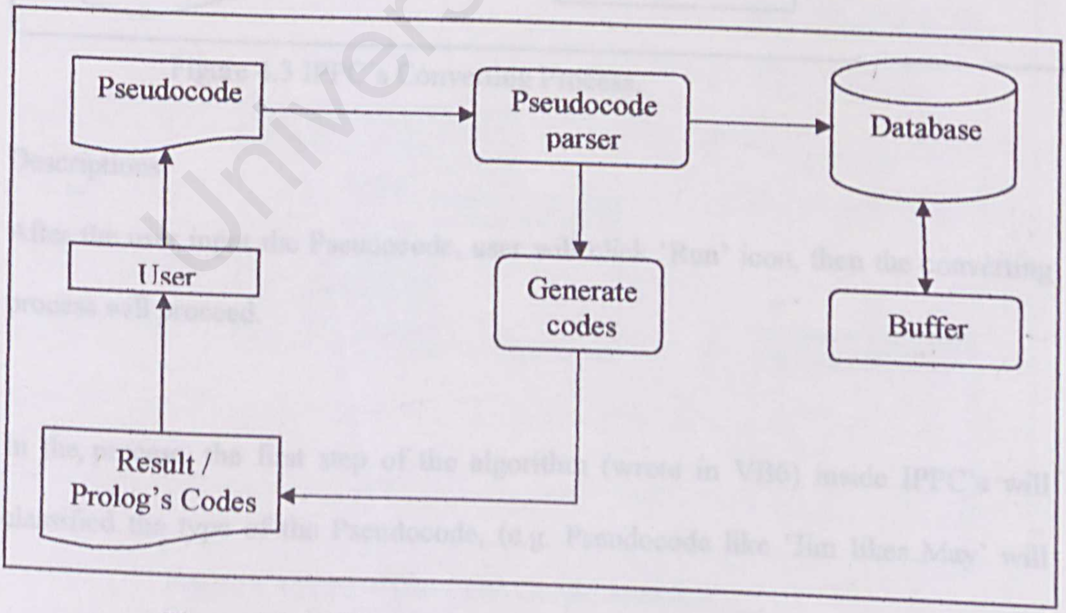


Figure 4.2 IPPC's Logical Architecture.

4.2.2 Level 2 – IPPC's Converting Process.

In the IPPC components, it need user to input the Pseudocode, thereby, IPPC must have an effective algorithm to convert the Pseudocode into Prolog codes. Below will give illustrate how the IPPC's converting process start and finish (refer Figure 4.3).

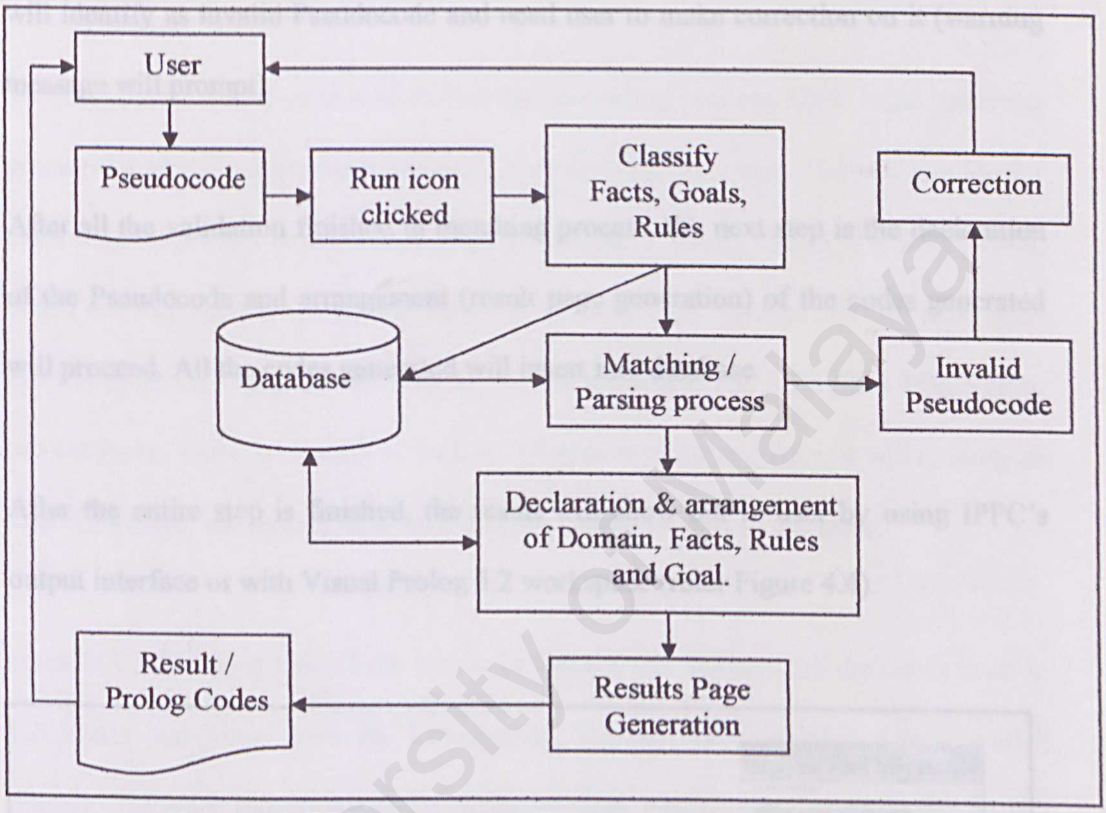

User
Pseudocode
Run icon clicked
Classify Facts, Goals, Rules
Correction
Database
Matching / Parsing process
Invalid Pseudocode
Declaration & arrangement of Domain, Facts, Rules and Goal.
Result / Prolog Codes
Results Page Generation

Figure 4.3 IPPC's Converting Process.

Descriptions:

After the user input the Pseudocode, user will click 'Run' icon, then the converting process will proceed.

In the process, the first step of the algorithm (wrote in VB6) inside IPPC's will classified the type of the Pseudocode, (e.g. Pseudocode like 'Jim likes May' will classify as Facts). After the Pseudocode was identified, it will insert the type into database.

The second step, matching of the Pseudocode will proceed. Matching process would succeed is depend on the words inside the database. If unmatched Pseudocode met, it will identify as invalid Pseudocode and need user to make correction on it (warning message will prompt).

After all the validation finished in matching process, the next step is the declaration of the Pseudocode and arrangement (result page generation) of the codes generated will proceed. All the codes generated will insert into database.

After the entire step is finished, the result will show out to user by using IPPC's output interface or with Visual Prolog 5.2 workspace (refer Figure 4.4).

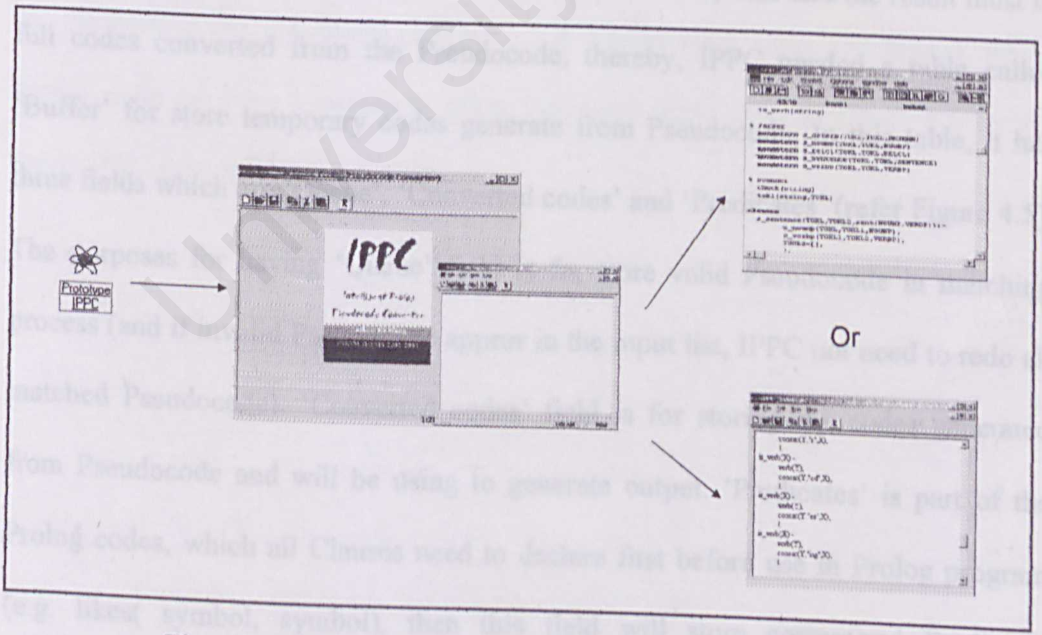


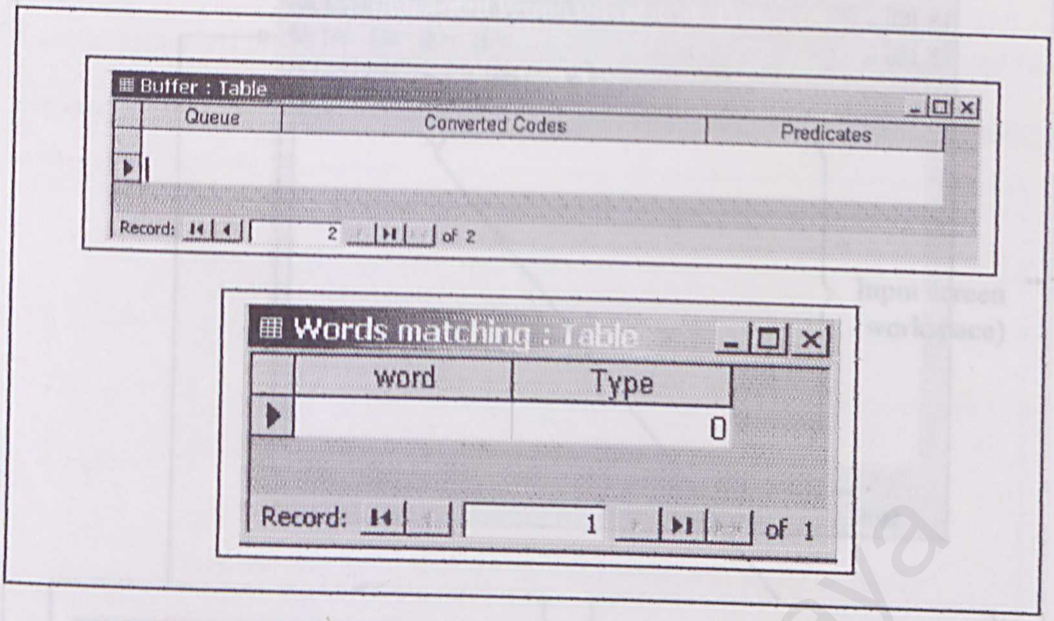
Figure 4.4 IPPC's Input and Output interfaces.

4.2.3 Level 3 – IPPC's Database Structures Design.

In this thesis, usage of database is required for the IPPC's Converting Process. There are a lot of types database, but for the tool, Microsoft Access is use as IPPC database. IPPC's Database consists of two tables – buffer (store result) and words for matching process.

To enable valid codes generated during the converting process, IPPC need matching process for identify arguments and predicates for a fact or a rule. Thereby, a table for store some important words was needed. In this table, words like is, are, were, verbs, nouns and correspond words will insert in a field called 'Words'. Beside that, from the words given, types of Pseudocode will be specified, e.g. 'Jim likes May', from this sentence 'likes' is classify as fact, or 'if' appear in the sentence, it will classify as rule. Therefore, the second field called 'Type' could be part of the table.

Because of the Pseudocode from user is in form of line-by-line and the result must in full codes converted from the Pseudocode, thereby, IPPC needed a table called 'Buffer' for store temporary codes generate from Pseudocode. In this table, it has three fields which are 'Queue', 'Converted codes' and 'Predicates' (refer Figure 4.5). The purposes for having 'Queue' field is for store valid Pseudocode in matching process (and if invalid Pseudocode appear in the input list, IPPC not need to redo all matched Pseudocode). 'Converted codes' field is for storing the codes generated from Pseudocode and will be using to generate output. 'Predicates' is part of the Prolog codes, which all Clauses need to declare first before use in Prolog program (e.g. likes(symbol, symbol), then this field will store correspond Predicates generated from IPPC.



Buffer : Table

Queue	Converted Codes	Predicates

Record: 2 of 2

Words matching : Table

word	Type
	0

Record: 1 of 1

Figure 4.5 IPPC's Database Structures.

4.2.4 Level 4 –IPPC's Graphic User Interface Design (A).

In this level, tool interface is taking into high measurement. IPPC's interface must be user friendly and easier to use. Therefore, IPPC's interface does not have command for performing the converting process but it creates from icons and menus, which give a more adaptive interface to users. For more detail refer to Figure 4.6 for the entire IPPC interface.

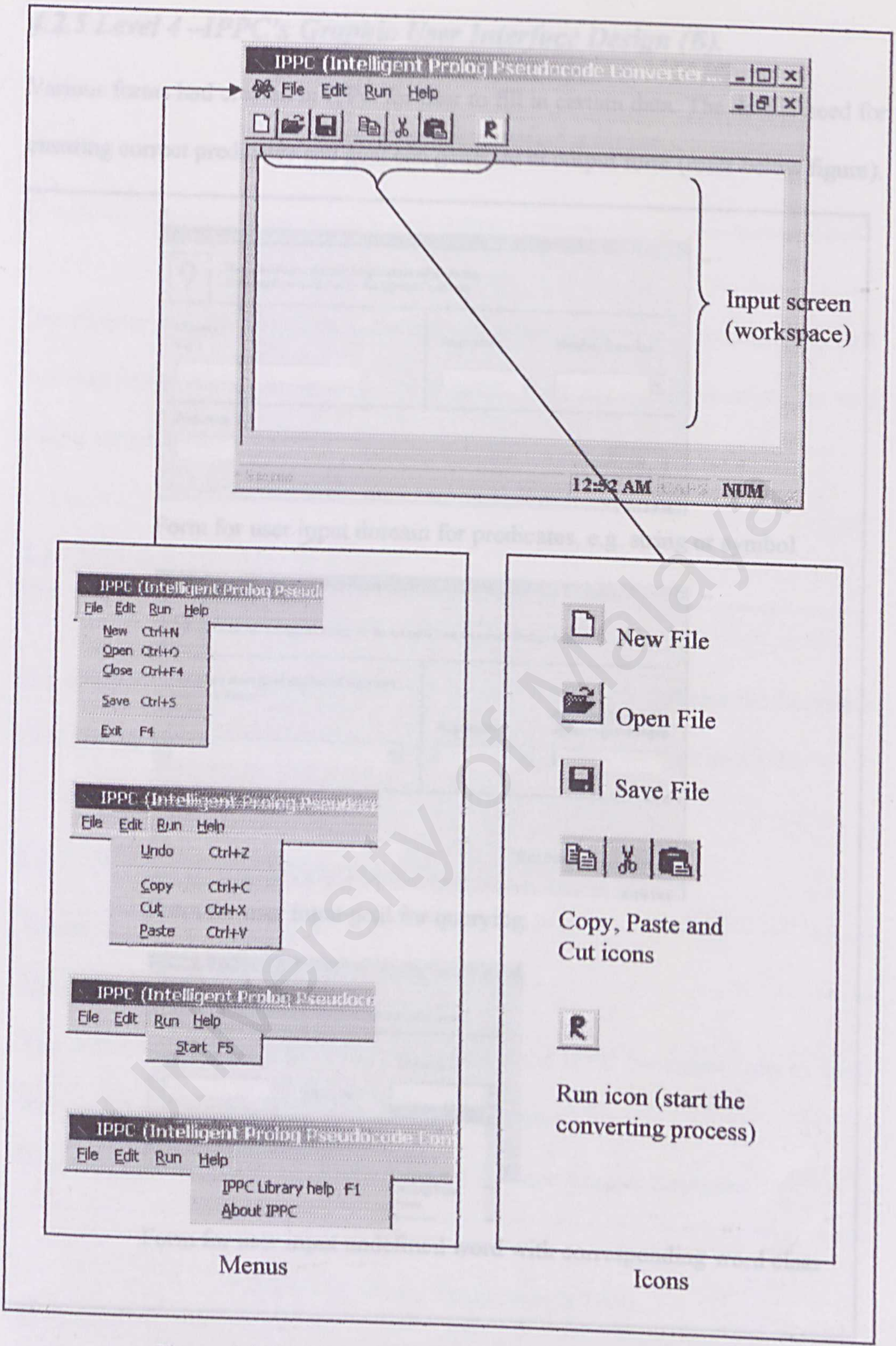
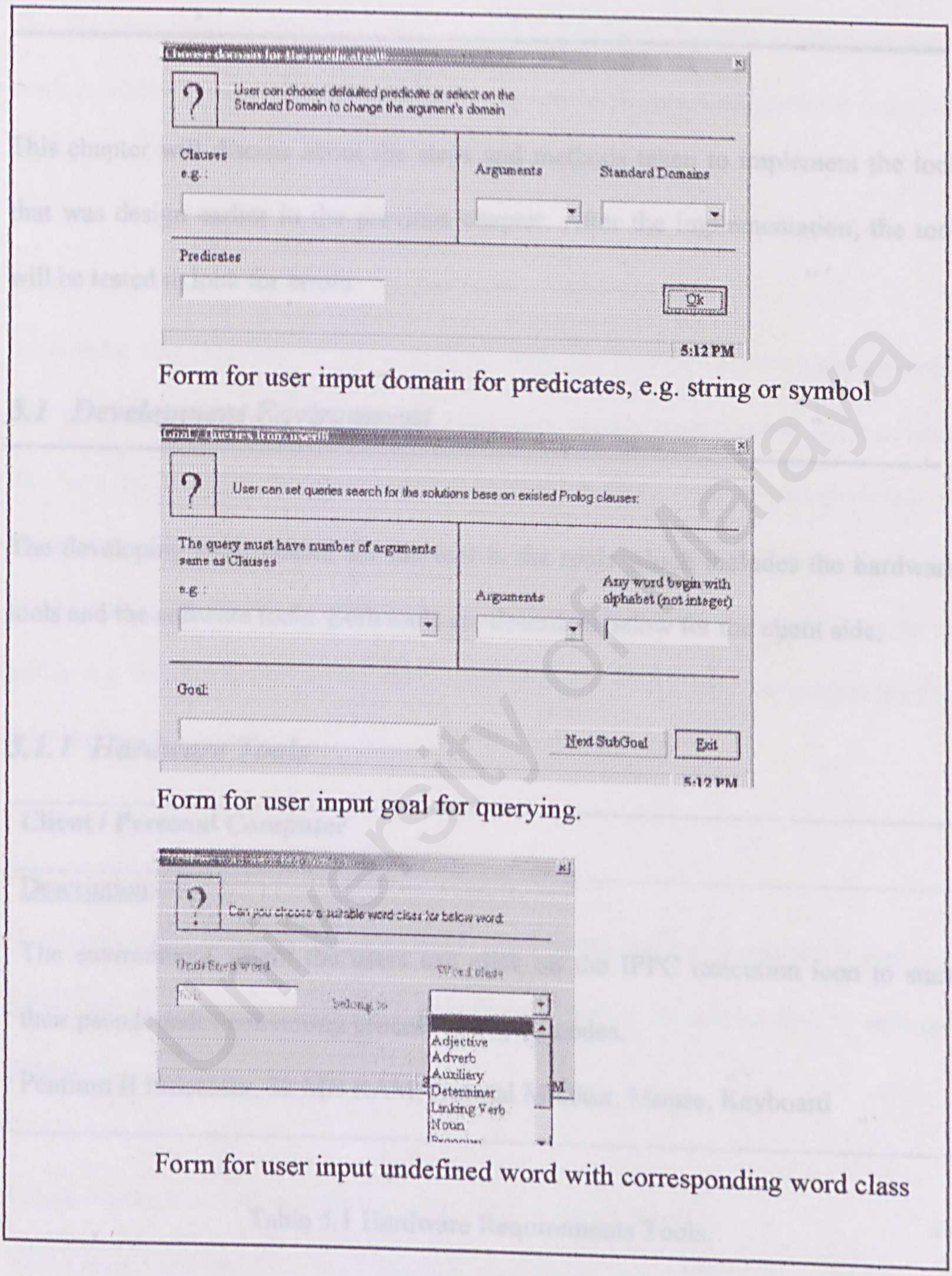


Figure 4.6 IPPC's Graphic User Interface (a).

4.2.5 Level 4 –IPPC's Graphic User Interface Design (B).

Various forms had created in IPPC for user to fill in certain data. The data is need for ensuring correct predicates and goal can generate in output form (refer below figure).



User can choose defaulted predicate or select on the Standard Domain to change the argument's domain

Form for user input domain for predicates, e.g. string or symbol

User can set queries search for the solutions bese on existed Prolog clauses:

The query must have number of arguments same as Clauses

Form for user input goal for querying.

Can you choose a suitable word class for below word:

Form for user input undefined word with corresponding word class

Figure 4.7 IPPC's Graphic User Interface (b).

Chapter Five

System Implementation

This chapter will discuss about the steps and methods taken to implement the tool that was design earlier in the previous chapter. After the implementation, the tool will be tested to look for errors.

5.1 Development Environment

The developing environment for the tool is the tools which includes the hardware tools and the software tools. Both tools are described below for the client side.

5.1.1 Hardware Tools

Client / Personal Computer
Description: The environment where the users can click on the IPPC execution icon to start their pseudocodes converting process to Prolog codes. Pentium II Processor, 32 MB RAM, Normal Monitor, Mouse, Keyboard

Table 5.1 Hardware Requirements Tools.

5.1.2 Software Tools

Windows OS -98, 2000, XP

The platform used to run the tool. The tool is platform dependent where it can't work in other OS which cannot support "exe" and "dll" files. This platform provided friendly interface in presentation and during debugging.

Visual Basic 6 and Visual Sourcesafe 6

It provides the easy way to develop IPPC with it's includes all intrinsic controls, plus grid, tab, and data-bound controls, and user friendly environment. A lot of time has been saved during the interface development because VB6 provided variety of templates for creating common application components. Provided integration with Microsoft Access in data retrieval or saving data. Visual Sourcesafe 6 has give an easier way to recover previous source code and provided a way for programmer to save their code among several computers.

Microsoft Access – 2000, XP

It provide programmer to upgrade words data in PrologData database. It provides good security in disable unknown person from changing the existed data by setting a password.

Visual Prolog 5.2 Personal Edition (free trial or full version)

VP5.2 is downloaded from the Prolog Development Center. This will enable the tool make compilation on the result generated from the tool.

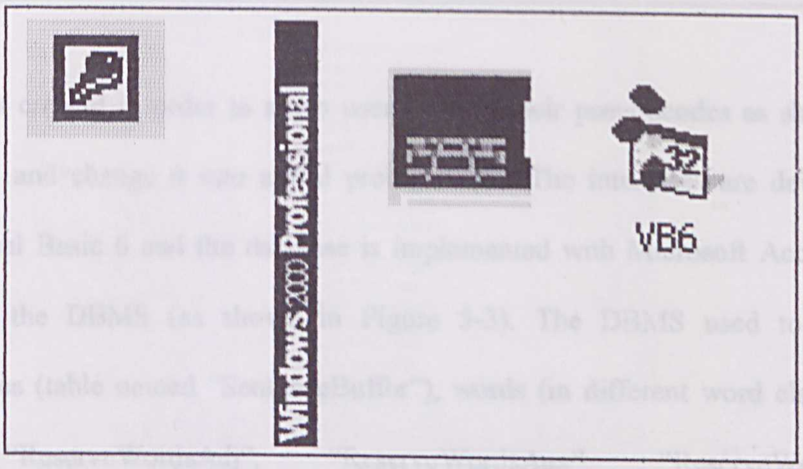


Figure 5.1 Software Tools.

5.2 *Tool Implementation.*

Implementation comprises of the system design structure to a computer readable system. The tool will be evolved from scratch design to a run able application. There are several implementations for this tool.

5.3 Interface and Database Implementation.

The tool is created in order to allow user keys in their pseudocodes as shown in Figure 5-2 and change it into actual prolog codes. The interfaces are developed using Visual Basic 6 and the database is implemented with Microsoft Access XP (2002) as the DBMS (as shown in Figure 5-3). The DBMS used to stored pseudocodes (table named "SentenceBuffer"), words (in different word classes in tables "ReserveWordsAdj", "ReserveWordsAux", "ReserveWordsN", "ReserveWordsV"), converted sentences structure (table named "NLPBuffer") and converted prolog codes (tables named "BufferCodesClauses", "BufferCodesDomain", "BufferCodesPredicates" and "BufferCodesGoal").

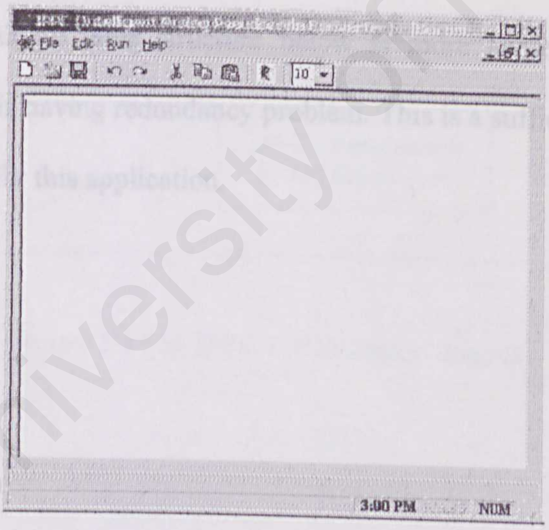
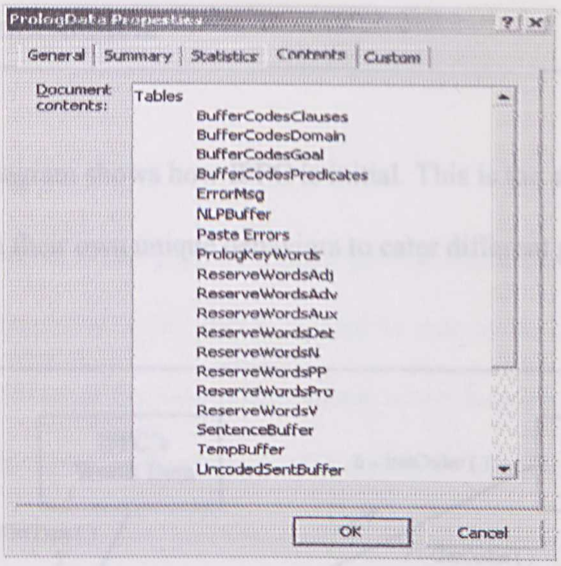


Figure 5.2 The IPPC's main page.



Document contents:

Tables
- BufferCodesClauses
- BufferCodesDomain
- BufferCodesGoal
- BufferCodesPredicates
- ErrorMsg
- NLPBuffer
- Paste Errors
- PrologKeyWords
- ReserveWordsAdj
- ReserveWordsAdv
- ReserveWordsAux
- ReserveWordsDet
- ReserveWordsN
- ReserveWordsPP
- ReserveWordsPro
- ReserveWordsV
- SentenceBuffer
- TempBuffer
- UncodedSentBuffer

Figure 5.3 IPPC's Database tables.

Do take note that the implemented table structure has changed from the proposed table structure at Figure 4.5 in Chapter 4: System Design. After the proper examination, the single table structure below is more appropriate because the proposed structure is having redundancy problem. This is a sufficient design to cater any data matching for this application.

5.4 IPPC Implementation.

The below UML diagram shows how IPPC is initial. This is the skeleton for each of the tool except with their own unique behaviors to cater different given tasks.

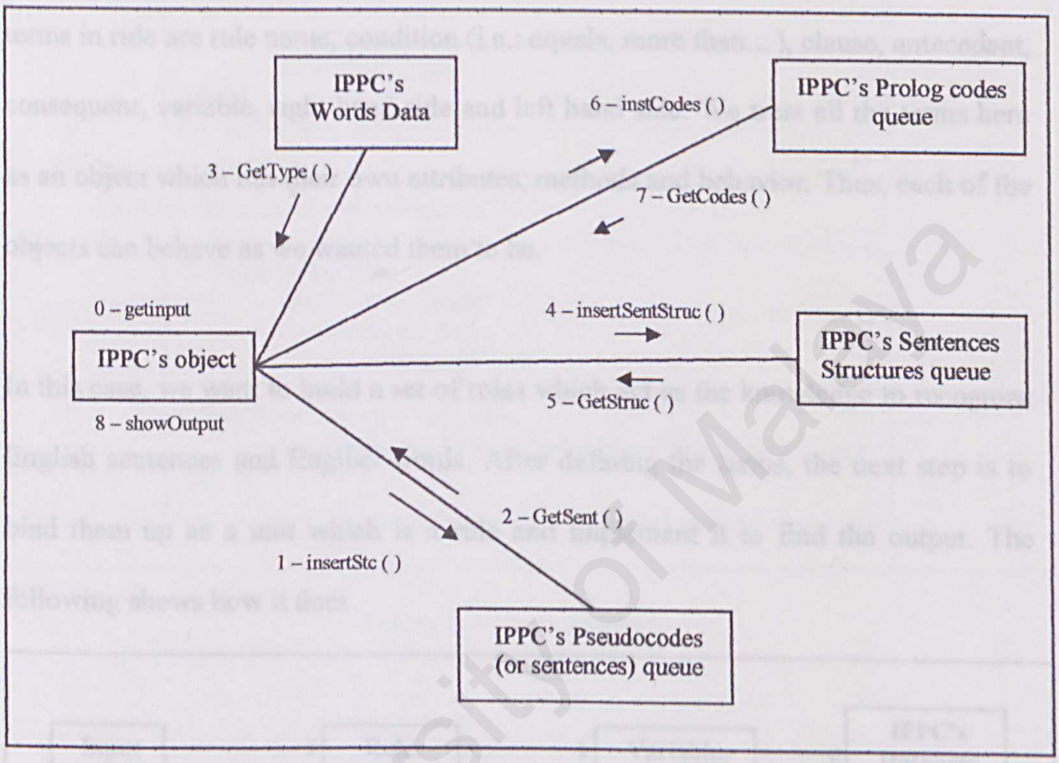


Figure 5.4 The IPPC collaboration diagram.

5.5 *Rules Implementation.*

5.5.1 *Using Rules in IPPC*

To represent rules as the knowledge in using object-oriented approach does making sense. We should treat everything which related to rule as an object. The general terms in rule are rule name, condition (i.e.: equals, more than...), clause, antecedent, consequent, variable, right hand side and left hand side. We treat all the terms here as an object which has their own attributes, methods and behavior. Thus, each of the objects can behave as we wanted them to be.

In this case, we want to build a set of rules which act as the knowledge to recognize English sentences and English words. After defining the terms, the next step is to bind them up as a unit which is a rule and implement it to find the output. The following shows how it does.

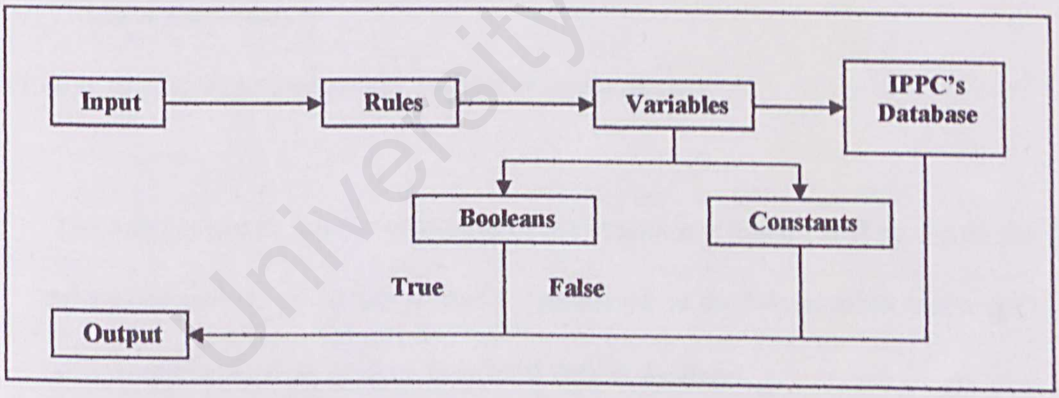


Figure 5.5 The diagram shows how to implement rules in the IPPC.

If the verb phrase have linking verb and noun then it be represent as sentence structure for verb phrase, else user must choose the next word class for create a sentence structure.

```
If WordTypeV = "lv_n" Then
        WordTypeV = "lv_n_"
        VP = WordTypeV
Else
        UndefWord = Word
        UsrDefWrdList.Show vbModal

        WordTypeV = WordTypeV & GetDefClsType & "_"
        VP = WordTypeV
End If
```

If the sentence is condition sentence with "If-Then" then do the "then" part again, else finish for the sentence structure generation.

```
If RuleEnabled Then
        CheckTxt = ThenTxt
        RuleEnabled = False
        GoTo TraceAgainIfThen
ElseIf RuleEnabled = False Then
        SentStructure = WordTypeIf & WordType & WordTypeV
End If
```

Figure 5.6 Rules example.

5.6 *Bottom-up Implementation.*

After all the needed rules were defined, it is considered that the tool are having a set of rules about the English sentences matching and Prolog codes constrains. The rules were coded in bottom-up method for produce sentences structure and prolog codes.

Bottom-up strategy is start with the words in the sentence and use rewrite rules backward to reduce the sequence of symbols until it consist solely of S []. For IPPC, bottom-up strategy is used until a full sentence structure is create and rewrite to major phrase like NP or VP or S will not proceed in IPPC.

When the IPPC get input from user, the IPPC's algorithm will goes as follow:

a) *Load a sentence from database.*

b) *Tokenize the sentence.*

c) *Use rules to match word with word class in database.*

The rules is use to justify what kind of the sentence structure will be. From the sentence, each word in the sentence (generated using tokenization technique) will matching with word class in existed data in database.

From the generation of sentence structure, bottom-up strategy is used. Figure 5.7 will describe more detail about how to apply this technique in IPPC.

d) *Insert the result into corresponding database.*

e) *Repeat a-d for other sentences and prolog codes routine.*

Sentence from database called "SentenceBuffer": John ate the cake.

From this sentence, the sequence of rewrite will look like this:

⇨ John ate the cake.
⇨ N ate the cake.
⇨ N V the cake.
⇨ N V Det cake.
⇨ N V Det N

The sentence structure is creating base on how the rule is coded.

Figure 5.7 Applying Bottom-up strategy in IPPC.

Above algorithm can be identified as a parsing algorithm, which means as a procedure for searches through various way (but in IPPC Bottom-up) of combining grammatical rules to find a combination that generates a tree that could be the structure of the input sentence [].

5.7 ActiveX Data Objects Implementation.

A cross-language technology for data access that exposes an object model incorporating data connection objects, data command objects, Recordset objects, and collections within these objects. The ADO object model provides an easy-to-use set of objects, properties, and methods for creating script that accesses data in databases []. IPPC had implemented Microsoft Access as database, thereby; some command codes are needed to make connection with it. Visual basic 6 have provided ADO for this solution. Below have shown some ADO's commands for using in connect the MS Access database with IPPC.

```
Private WithEvents connConnection As ADODB.Connection
Private WithEvents rsInfo As ADODB.Recordset

    .....
    strProvider = "Provider= Microsoft.Jet.OLEDB.4.0;"
    strDataSource = App.Path
    strDataBaseName = "\PrologData.mdb;"
    strDataSource = "Data Source=" & strDataSource & _
    strDataBaseName
    strConnect = strProvider & strDataSource

    Set connConnection = New ADODB.Connection
    connConnection.CursorLocation = adUseClient
    connConnection.Open strConnect
    .....
```

Figure 5.8 ADO's Commands.

5.8 Exception Handling Implementation.

Ever since the beginning of programming languages, error handling is one of the most difficult issues. An exception is a n object tat is "thrown" from the site if the error and can be "caught" by an appropriate exception handler designed to handle that particular type of error. VB6 had provided exception handling for error detection and recover from bad situation.

```
On Error GoTo ErrorHandler
    .....

    .....
ErrorHandler:
    MsgBox "GetSent - part_3SQL (part_3.cls)", vbCritical
```

Figure 5.9 Exception Handling method.

Chapter Six

System Testing

Software testing is one of the main phases in the Waterfall Life Cycle model. In this phase, the process of testing and debugging are done to detect defects and bugs of the tool - IPPC. These processes are usually done incrementally with system development.

This phase is also often referred to as Verification and Validation (V & V). Verification refers to the set of activities that ensure the software correctly implements a specific function. Validation refers to a different set of activities that ensure the tool has been built is traceable to user requirements. A successful test is one in which no errors are found.

The objectives to test this tool are:

a) To reveal rules error based on the simple English sentence structure.

b) To compare the expected outcome with the actual outcome. Eventually, debug it to enhance it functionality and capability.

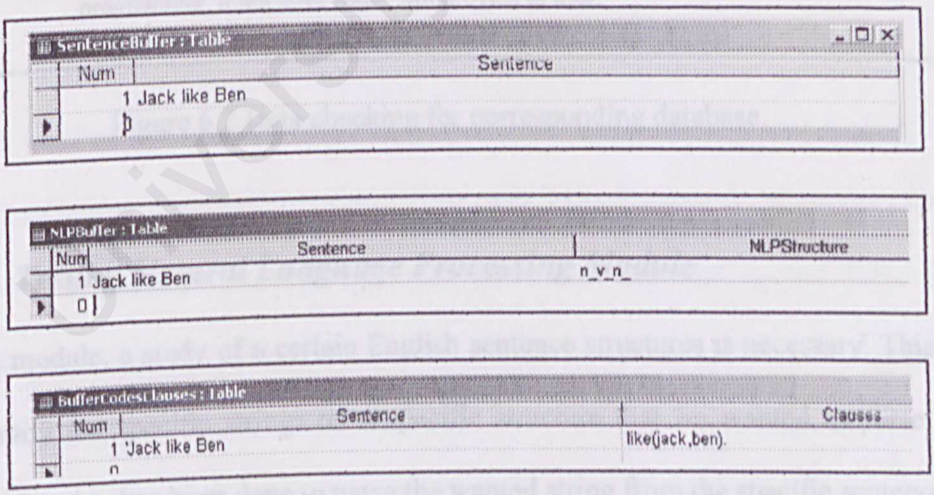
c) To ensure the Prolog code created by IPPC and has been shown to users and according to the user's Pseudocodes.

6.1 Unit Testing

Unit testing focuses verification effort on the smallest unit of tool design corresponding to the tool components or modules.

6.1.1 Testing Display and Database Module

This is referring to the IPPC interface and database testing to ensure that every data which user insert into the interface is stored accurately and correctly to the correspondence database. A set of sample raw data is created for the testing purpose in this module. The process includes iterate the checking on the duplicated data in database to ensure that every entered data is valid and ease the redundancy and duplication problem. The control objects such as radio button, shortcut icons, combo box and text field are tested too to ensure the correct functionality respectively.



SentenceBuffer : Table

Num	Sentence
1	Jack like Ben
0	

NLPBuffer : Table

Num	Sentence	NLPStructure
1	Jack like Ben	n_v_n
0		

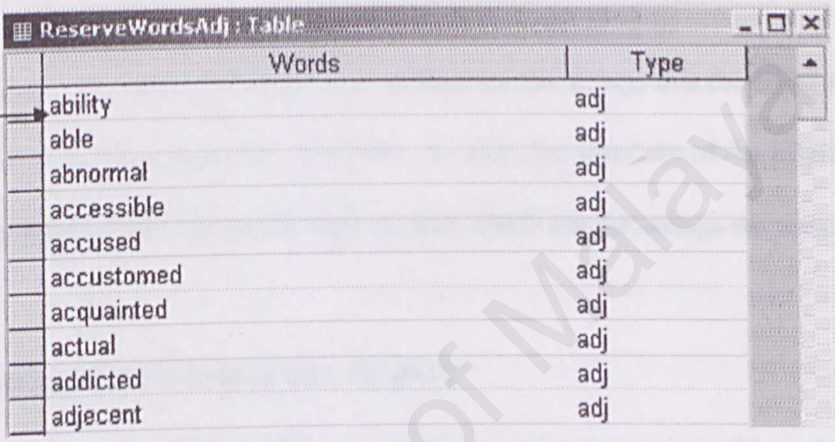
BufferCodesClauses : Table

Num	Sentence	Clauses
1	Jack like Ben	like(jack,ben).
0		

Figure 6.1 Testing data insert into correct table.

6.1.2 Checking Database Data

A set of words for nouns, verbs, auxiliaries, determiners, and adjectives data is prepared for IPPC during the user need to convert the Pseudocodes. Then, the words in the pseudocodes are randomly checked to generate the correct sentence structure based on those data. Manually checking all of the data with correct word class are done by programmer.



ReserveWordsAdj : Table

Words	Type
ability	adj
able	adj
abnormal	adj
accessible	adj
accused	adj
accustomed	adj
acquainted	adj
actual	adj
addicted	adj
adjecent	adj

Ability is not a "adj" but is "n". Error detected and must be correct by programmer, if not error result will provide to user.

Figure 6.2 Data checking for corresponding database.

6.1.3 Testing Natural Language Processing Module

In this module, a study of a certain English sentence structures is necessary. This is to identify the specific strings on a specific structure that we wanted to parse. A serial of testing has been done to parse the wanted string from the specific sentence.

NLP simple context grammar:

⇨ S => NP VP
⇨ NP => ART ADJ N
⇨ NP => ART N
⇨ NP => ADJ N
⇨ VP => AUX VP
⇨ VP => V NP

Must include in IPPC.

Figure 6.3 Testing for word parsing of a sentence.

The "art / det", "n", "adj", "v", and "aux" strings are the strings that the IPPC should parse to the words for a sentence. Therefore, to study the structure of the English for IPPC is essential in order to get the right results, which are the strings we want.

6.1.4 Testing Converting Rules Module

Testing has been done on the bottom-up strategy to ensure that it could produce the desired result. This includes preparing a set of data.

To test whether the rules are working or not, we must make sure that each of the rules have return the correct sentence structure as set in Figure 6.3. When the rule is fire, say "if wordtype = "det"" the then part must constrain to find following sentence structure like: det_adj_n or det_n. If the rules unable to give the desirable structure, programmer must make correction on it. Else, there's a bug. Hereby, we can say that the level of result gave has been decrease. Therefore, adding accurate rule will enrich the IPPC reliability and vice versa.

To get the sentence structure: Det_Adj_N, below rules must satisfy.

```
⇨ WordType = WordType & turn2SQL.GetType(Word, Det)
⇨ .....
⇨ If WordType = "det" Then
⇨ .....
⇨ WordType = WordType & "_" & turn2SQL.GetType(Word, Adj)
⇨ .....
⇨ If WordType = "det_adj" Then
⇨ .....
⇨ WordType = WordType & "_" & turn2SQL.GetType(Word, N)
⇨ .....
⇨ If WordType <> "det_adj_n" Then
⇨ .....
```

Figure 6.4 Samples of rules.

6.1.5 Testing Prolog Codes Generation Module

Prolog codes were the main purpose of implementing IPPC. Thereby, algorithm for generate prolog codes must be test. In this module, codes for generate predicate and clauses will give accurate result and executable in VP5.

6.2 Integration Testing

For this project, a bottom-up approach has been used. Bottom-up integration testing begins construction and testing with modules at the lowest levels of the system and then moving upward to the modules at the higher levels of the tool.

Since all the modules have been tested and have been declared bug free, the modules will combine one by one according to the called modules. The integration is checked again to ensure there is no error.

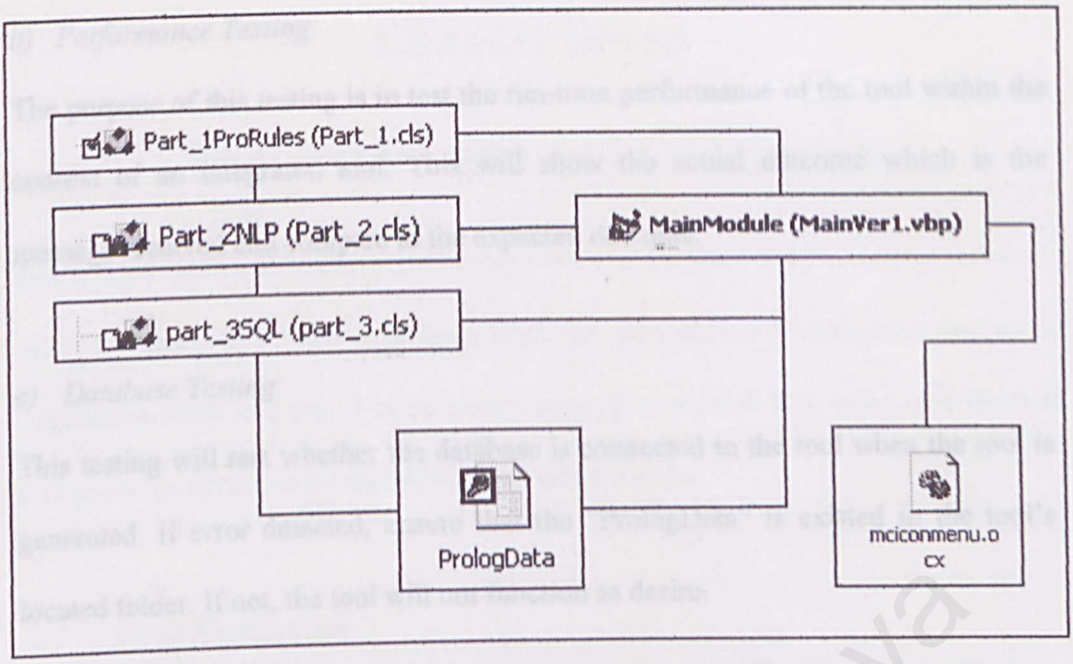


Figure 6.5 Illustration of tool integration.

6.3 Tool Testing

Tool testing is a series of different tests designed to fully exercise the tool to uncover its limitations and measure its capabilities. The objective is to test an integrated tool and verify that it meets specified requirements.

There are several types of system testing that are worthwhile for the tool - IPPC.

a) Rule Testing

It is a test during the run time environment where the complete set of rules is loaded. If the pseudocodes (or sentence) converted into prolog codes, then it has considered success.

b) Performance Testing

The purpose of this testing is to test the run-time performance of the tool within the context of an integrated tool. This will show the actual outcome which is the message received and compare to the expected outcome.

c) Database Testing

This testing will test whether the database is connected to the tool when the tool is generated. If error detected, ensure that the "PrologData" is existed in the tool's located folder. If not, the tool will not function as desire.

6.4 Testing Analysis

Overall, the tool runs smooth. All of the codes and words transformation work well in database and execution files. IPPC is able to generate correct result. As a conclusion, all the objectives have been achieved.

Chapter Seven

Tool Evaluation and Conclusion

After the tool – IPPC is successfully built; the next phrase is evaluation and some conclusion about this tool. The necessary need of evaluation is to make sure the tool met goals for performance and other desirable attributes.

7.1 Tool Evaluation

During the period of coding and implementation of this system, various problems were encountered. These problems were solved through research and studies in fields such as the Natural Language Processing (NLP), Prolog Programming, Visual Basic 6 programming, journals and reference book. The system's strengths, limitations, and future enhancement were identified.

7.2 Problems Encountered And Solutions

Problems are everywhere and so do in every thesis. Several problems encountered throughout the development of this tool. These include:

a) Difficulty in Choosing Development Technology and Tools

There are many software tools available to develop an IPPC. Choosing a suitable technology and tools was a critical process as all tools possesses their own strengths and weaknesses. In addition, the availability of the required tools for development was also a major consideration. Eventually, VB6 has been chosen of its strength in ADO and ease interface creation. We only have to learn about the methods to program rather than create all the codes by ourselves especially hard interface coding. It saves us a lot of time.

b) Lack of Knowledge applying Natural Language into IPPC

Theoretical has never been hard. All the problems and doubts will emerge once the development and design starts. It's hard to find a good linguistic book which provides full knowledge about Natural Language processing. Without good explanations of NLP, the sentence structure for the IPPC and the rest of the process are hard to proceed. For this problem, some simple grammar was taken for the tool and for complex grammar will not include in this thesis.

c) Lack of Knowledge in Visual Prolog

Previous knowledge in traditional, such as C, Java or VB6 is more to object oriented programming, but Visual Prolog is a new or non-popular development programming which have brought uncomfortable to programmer. Since there was no prior knowledge of programming in Visual Prolog, there was an uncertainty on how to organize the codes into the tool. These new programming languages and concepts were never taught before and to implement such as application requires a fair grasp of the languages. These programming approaches seem to be totally different from the traditional programming languages.

d) Sentence structure (Grammar) Parsing Problems

There are varieties of grammar. There have simple to complex grammar representation. Arrangement between words in a sentence can give different meanings. This makes the parsing from the sentence into sentence structure becomes more predictable.

e) Tool Testing Problem

As we know, the sentences matching are not familiar to student especially for student not from linguistic domain. But I need to test my tool very frequently for make sure all the desired goal met in IPPC. Therefore, I had to create prototype IPPC executable program to run the test locally on the PC before I can completely proving it safe from bugs.

7.3 Tool Strengths

During the development of this project, several tool strengths were identified and described as follow:

a) User Friendliness and Easy to Use Interface

Some useful Graphical User Interface (GUI) such as shortcut buttons, check boxes and drop-down list boxes are created on the IPPC which could attract the users to navigate through the tool and give faster access. This user-friendly interface can minimize learning time for the user.

b) Database Access Authority

All the data are organized and stored in the form of database using Microsoft Access 2000. It is a data stored database and any changes can make to the error. So for security, it was needed password to open it for modification.

c) Semi-Autonomous

After the user activated the tool to generate Prolog codes, the user can leave the rest of the task to the tool. The tool will do parsing algorithm consider to the rules. If there have problems like undefined word, or error detected IPPC will pop out a message to user about the problem. User need to make choice depend on the pop from. The last result will give to user if problem in the converting process.

d) Bottom-up strategy and rules

The selected strategy enables the searching for the words in the sentence and changing it into corresponding word class. Every piece of constructed rule acts as the part of the knowledge and the rules can lead to the correct generation of prolog codes.

e) Ease to upgrade for new version

IPPC is built with VB6 which provide programmer to separate the functionality and execute module in different projects. Most of the NLP and Prolog code generation codes are placed in '.dll' file. The programmer needs to change the '.dll' file part without affect the main module (interface).

7.4 Tool Limitations

Owing to the time constraint and the programming language itself, there were some limitations in this tool. These include:

a) IPPC is limited in English

IPPC mainly built for convert Pseudocodes composed with English word (but can upgrade to Malay).

b) Unable to detect words other than Noun, Verb, Adjective and Possession Noun.

IPPC is built with simple grammar (sentence structure). A sentence is construct with Noun Phrase and Verb Phrase (S = NP + VP). Preposition words is not included in IPPC (can upgrade in new version).

- S = NP + VP
- NP = N or Adj + N or Det + Adj + N or N's N
- VP = V or V + NP or Aux + V or Aux + V + NP or Aux + NP

c) The sentence must be in correct grammar form.

The user must key in correct sentence structure as defined in IPPC. If user do not follow that constraints, IPPC will be prompt out message "invalid sentence" or give incorrect result.

d) Simple prolog codes generate by IPPC.

In Prolog, there are few major part needed to generate a executable program. Part of them is Clauses, Predicates and Goal (Domains not included in IPPC). Because of recently IPPC can convert simple sentence structure (as shown above), so the prolog codes generate by IPPC also is simple codes. User can't get complex rules or clauses from IPPC, e.g. list (its need conjunction and more parsing algorithms).

e) IPPC input cannot more than 7 words for fact's sentence and cannot more than 16 words for rule's sentence.

7.5 Tool Enhancement

Further development and many new ideas have come about while the tool was being implemented. Owing to time constraint and other factors, not all of the ideas could be incorporated into IPPC. It is hoped that the following aspects could be considered in future:

a) Adding more sentence structure with preposition and adverb.

Since the IPPC parses only noun, verb and adjective of the pseudocodes but IPPC still can extend to more useable tool with preposition, it is possible to implement preposition into IPPC. Thus, it is able to parse for the meaning of the sentence into more complex prolog codes, e.g. time, person or data in list format.

b) Matching technique must be change into more comfortable technique

In order to make the IPPC more intelligent, it may check on the matching algorithm. Because of recently technology can't well understand the meaning of human act or human languages. Thereby, if have more efficient technique has create for this purpose, it can apply into this tool for improve it performance.

7.6 Conclusion

In this paper, several chapters have clearly described about the tool, IPPC. In process to starting, doing and finishing this paper, much kind of problems, knowledge and experiences had gained from this paper.

IPPC still not complete call as intelligent tool but it still can generate few types of prolog codes from user's pseudocodes. Thereby, it must be improve to accomplish the goal.

Finally, the results show that recently technology is needed to improve to more intelligent behavior for ease human to apply human act or human language straightly into high technology without need of this converter. While this study investigate differential view of computer software and human behavior, it give another idea for students to learn more on how to managing and researching a project.

Bibliography

[1] Ivan Bratko, PROLOG - Programming for artificial intelligence, Third edition, Addison Wesley, 2001.

[2] Paul Brna, Prolog Programming, March 5, 2001. http://www.cbl.leeds.ac.uk/~paul/prologbook/brna.pdf.gz

[3] Visual Prolog Version 5.0, Language Tutorial, Borland International, 1988.

[4] Krassimir Yalumov, Artificial intelligent II: Methodology, systems, applications, 1987. The Nesy Prolog, Pg. 63-68.

[5] PERDRIX Herve, Artificial intelligent II: Methodology, systems, applications, 1987. Program synthesis from specifications, Pg. 13-20.

[6] James Allen, Natural Language Understanding, Benjemin / Cummings, 1987.

[7] Fischler, Martin A., Intelligent: The eye, the Brain, and the Computer, Addison-Wesley, 1987.

[8] Clicksin, W.F & Mellish C.S, Programming in Prolog, 2nd edition, Springer-Verlag Berlin Heidelberg 1981, 1984

[9] Computer User, High-tech Dictionary, http://www.computeruser.com/resources/dictionary/index.html

[10] MSDN Library Visual Studio 6.0 release, 1991-1998 Microsoft Corporation.

[11] Nexus Internet Solutions Ltd. , 2000 http://www.nexusdesign.com/default.asp

[12] File Extensions Windows/OS2/Apple/UNIX, http://www.icdatamaster.com/index.html

[13] Textproc,
http://centaurs.mtk.nao.ac.jp/~shiki/Comp/Usage/PortsIndexMaker/textproc.html
http://www.ctgi.net/nicetext/index.html

[14] International Conference on Information and Computer Security 1997 (ICICS97) in Beijing, China back in November, 1997.

[15] Introduction of Microsoft Access, copyright 2001.
http://www.functionx.com/access/

[16] Shary Lawrence Pfleeger, Software Engineering: Theory And Practice, 2nd Ed, Practice Hall, 2000.