An Effect of Particle Swarm Optimization on SDLC

Shrishti Tamrakar (M.Tech Scholar),
Anubhav Sharma (Asst.Prof.)
Acropolis Institute of Technology and Research, Bhopal

Abstract— Software Engineering is a systematic approach to the event, maintenance, and retirement of the software system. The Software requirement specification is a technical specification of requirements for the software product. Formal specification is the cluster of statements expressed in a language whose vocabulary, syntax and semantics are formally defined. Formal specification focuses on covering unsolved faults that would probably go ignored with tradition specification methods. So, it leads to correct and bug free software. Many approaches are there for formal specification of software. This paper has used model-based approach which supports tools like B-notation, Z-notation, VDM, etc. As VDM has various advantages over Z and B, so Formal specification is done using VDM. As compiler is also software which performs many operations. As part of the research, memory allocation has been taken into consideration for formal specification. Whenever compiler allocates memory, fragmentation occurs. To optimize the allocation for reducing the internal and external fragmentation, particle swarm optimization algorithm is used and it is formally specified. Compiler memory allocation operation takes discrete values, so discrete particle swarm optimization should be specified. Using the VDM, the discrete particle swarm optimization algorithm is formally specified. Further, compiler memory allocation can be done using this algorithm; it will help for reducing the fragmentation. For implementing the VDM, overture tool is used.

Keywords: Formal Specification, Z, B, VDM, compiler, fragmentation, particle swarm optimization

I. INTRODUCTION

SOFTWARE ENGINEERING can be defined as “The systematic approach to the event, operation, maintenance and retirement of the software system”. The primary goal of software engineering is to boost the standard of software product. The analysis phase of software development involves project planning and software requirement definition. The software requirement specification is a technical specification of requirements for the software product. The goal of software requirement specification definition is to completely and consistently specify the technical requirements for the software product in a concise and unambiguous manner, using formal notations appropriate. The software requirement specifications based on the system definition. The requirement specification will state that the ‘what’ of the software product without implying ‘how’. For Software formal specification, formal methods are used which are mathematical-based techniques used for specification, proving and verification of software systems [1]. The process of formal verification means that applying these approaches to verify the properties making certain correctness of a system. Formal verification of software system targets the computer program wherever linguistics of the language provides precise aiming to the program analyzed. Formal specification can be used to provide an unambiguous and consistent supplement to natural language descriptions and can be rigorously validated and verified leading to the early detection of specification errors [2].

Fig.1 Formal Specification

Fig.1 depicts where the Formal Specification lies in the Software Development Life Cycle. It illustrates that between the User requirement Definition and the High level design there exists the Formal Specification. Its advantage is that it increases the contractor involvement rather than the client involvement.

Various researches have been made in different fields by using Z [17], B [11] and VDM [16] formal
methods. Almeida et.al [7] in 1992 transformed a semi formal specification to VDM. Descriptions of the requirements of a software system written in an unconstrained natural language are considered to be informal. Informal descriptions are known to have the potential to contain ambiguities, partial descriptions, inconsistencies, and incompleteness and poor ordering of requirements. Specifications written in VDM like language are considered formal. In between these two ends they recognized several techniques for semi-formal specifications. In this paper they proposed a technique for semi-formal specification.

Ledru [9] in 1993 developed a reactive system in a VDM framework. This paper studied detailed development of reactive systems, using an extension of VDM. The extension allows specification and proof of behavioral aspects to be expressed in the VDM framework. This is achieved by using traces of the input/output activities and introducing the notion of external entities whose behavior is described by a state machine. The major objective of this work is to improve understanding of the practical implications of the specification, design, and symbolic validation of machine-checked reactive systems.

Rania Hassan, et.al [24] in 2005 compared the genetic algorithm and particle swarm optimization. PSO is like GA in the sense that both are heuristic population-based search method. PSO and GA move from a set of points (population) to another set of points in a single iteration with likely improvement using a combination of deterministic and probabilistic rules. It is claimed that PSO gives the same effect as GA but it has significantly better computational efficiency by implementing statistical analysis and formal hypothesis testing.

Ponsard et.al [20] in 2006 analyzed formal requirement models to Formal Specifications in B. They analyzed that the development of critical systems requires a high assurance process from requirements to the running code. Formal methods, such as B, now provide industry-strength tools to develop abstract models refine them in more concrete models and finally turn them into code. A major remaining weakness in the development chain is the gap between textual or semi-formal requirements and formal models. In this paper, they explore how to cope with this problem using a goal-oriented approach to elaborate a pertinent model, including regulation modelling, and turn it into a high quality abstract formal specification.

Mojtaba Ahmadieh Khanesar, et.al [23] in 2009. In this research, a new interpretation for the velocity of binary PSO was proposed, which is the rate of change of bits of particles. The previous values, state and direction were considered that helped finding better solutions. They presented the application of Binary PSO where search space values are discrete in decision making, travelling salesman problem, solving lot size problem, etc.

Dantas [11] in 2009 presented verified compilation and the B method: A Proposal and a first appraisal. This paper investigates the application of the B method beyond the classical algorithmic level provided by the B0 sub-language, and presents refinements of B models at a level of precision equivalent to assembly language. It claim and justify that this extension provides a more reliable software development process as it bypasses the less trustable steps in the application of the B method: code synthesis and compilation. The results presented in this paper have a value as a proof of concept and may be used as a basis to establish an agenda for the development of an approach to build verifying compilers based on the B method.

Zafar et.al [3] in 2011 worked in Transformation of class diagrams into Formal Specification. He says that requirement analysis and design specification is a serious issue in software engineering because of semantics involved in the transformation of real world problems to computational models. Unified Modeling language (UML) has been accepted as a standard for design and development of object oriented systems. UML has a lack of notations for description of a complete functional system and its semantics is still semi-formal allowing ambiguities at desing level. Formal methods involve much mathematics. Therefore, a strong linkage of UML and formal methods is needed to overcome the above issues. In this paper, an integration of UML and Z notation is defined for class diagrams considering both the syntax and semantics at an abstract level of specification.

Buragga et.al [4] in 2011 analyzed formal parsing of CFG (Context-Free grammar) using Left most Derivations. Formal approaches are useful to verify the properties of software and hardware systems. Formal verification of a software system targets the...
source program where semantics of a language has more meanings than its syntax. Therefore, program verification does not give guarantee the generated executed source code is correct as described in the source program. This is because the compiler may lead to an incorrect target program due to bugs in the compiler itself. It means verification of compiler is more important than verification of a source program to be compiled. In this paper, context-free grammar is linked with Z notation to be useful in the verification of a part of compiler. Firstly they defined the grammar, then language derivation procedure is described using the left most derivations. Next, verification of a given language is described by recursive procedures. The ambiguity of a language is checked as a part of the parsing analysis. By reading all these literatures it is found that B and Z notations have been used mostly, but the same work can be done through VDM which will give better output and easy to be work with.

The paper is organized as follows: Section II provides the problem definition of informal and formal specifications like in Software industry where the problem lies in specifying the software and pros and cons of formal specification, and also describes ignorance of compiler efficiency. Section III presents the methodology used in specifying the particle swarm optimization by using VDM. Section IV reports the result of the formal specification of discrete particle swarm optimization. Finally, Section V draws preliminary conclusions on this result and an agenda for future research.

II. PROBLEM DEFINITION

1. Drawbacks of informal specification over formal specification:[25] It is often claimed that the use of formal specification languages gives rise to the desirable properties of clarity and abstraction in specifications, and that their use leads to the early detection of ambiguity and inconsistencies in requirements. The arguments for formal specifications resulting in clear and unambiguous specifications are that: formal notations have a mathematical semantics, and hence the meaning of a given specification need not be in doubt for it can be determined from the semantic model; and that the rich set of mathematical constructs available in formal specification languages, such as Z, and their declarative style, encourages an abstract statement of a system’s properties and hence results in specifications which are easier to understand as they are not cluttered with unnecessary implementation details. Furthermore, the production of a formal specification is supposed to result in the early detection of inconsistencies because its production encourages the systematic consideration of the requirements. The ability to perform a limited amount of type-checking automatically on formal specifications removes another source of inconsistency.

2. Ignorance of Compiler efficiency

Compiler performs many operations like lexical analysis, pre-processing, parsing, semantic analysis, code generation, code optimization, task scheduling, memory allocation, etc. Any of these if not formally specified with any one of the approaches, then it affects the compiler efficiency. Already lexical analysis, pre-processing, parsing, semantic analysis, all these have been specified using Z notation, but memory allocation and task scheduling comes under optimization process, so this needs to formally specify the optimization techniques first. In this paper, particle swarm optimization is used for scheduling and its formal specification has been done. Since compiler memory allocation operation takes discrete values, discrete particle swarm optimization is formal specified.

III. METHODOLOGY

1. The Vienna Development Method is a mature formal method whose origins go back to the IBM Vienna Laboratory in the 1970s. It is a formal method for the description and development of computer systems. Its formal descriptions (or ‘specifications’) use mathematical notation to provide a precise statement of the intended function of a system. Such descriptions are built in terms of models of an underlying state with a collection of operations which are specified by pre- and post-conditions. VDM designs are guided by a number of proof obligations whose discharge establishes the correctness of design by either data rectification or operation decomposition. Thus it can be seen that VDM addresses the stages of development from specification through to code. From the wide variety of tools available it single out the Overture
Automatic Proof System (APS) and the VDMTools for type checking, interpretation and code generation. Peter Gorm Larsen [10] presented “Recent industrial Applications of VDM in Japan”. He analyzed that there is an industrial use of VDM in Japan since the acquisition of VDMTools by CSK systems. This acquisition followed a very successful application of VDM++ in the development of two subsystems of the Trade one back office system for securities trading. FeliCa Networks also applied VDM++ in the development of a new generation IC chip for use as an electronic purse which can be embedded in a cellular telephone.

2. Particle Swarm Optimization (PSO) [24] is one of the evolutionary optimization methods inspired by nature which include evolutionary strategy (ES), evolutionary programming (EP), genetic algorithm (GA), and genetic programming (GP). PSO is distinctly different from other evolutionary-type methods in that it does not use the filtering operation- crossover and mutation and the members of the entire population are maintained through the search procedure. In PSO algorithm, each member is called “particle”, and each particle flies around in the multi-dimensional search space with a velocity, which is constantly updated by the particle’s own experience and the experience of the particle’s neighbors. Since PSO was first introduced by Kennedy and Eberhart (1995, 2001), it has been successfully applied to optimize various continuous nonlinear functions. Although the applications of PSO on combinatorial optimization problems are still limited, PSO has its merit in the simple concept and economic computational cost.

The main idea behind the development of PSO is the social sharing of information among individuals of a population. In PSO algorithms, search is a Binary Particle Swarm Optimization Algorithm conducted by using a population of particles, corresponding to individuals as in the case of evolutionary algorithms. Unlike GA, there is no operator of natural evolution which is used to generate new solutions for future generation. Instead, PSO is based on the exchange of information between individuals, so called particles, of the population, so called swarm. Each particle adjusts its own position towards its previous experience and towards the best previous position obtained in the swarm. Memorizing its best own position establishes the particle’s experience implying a local search along with global search emerging from the neighboring experience or the experience of the whole swarm.

Two variants of the PSO algorithm were developed, one with a global neighborhood, and other one with a local neighborhood. According to the global neighborhood, each particle moves towards its best previous position and towards the best particle in the whole swarm, called gbest model. On the other hand, according to the local variant, called lbest model, each particle moves towards its best previous position and towards the best particle in its restricted neighborhood. The discrete binary version of the PSO has already been developed. PSO has been successfully applied to a wide range of applications such as power and voltage control, mass-spring system, and task assignment. The comprehensive survey of the PSO algorithms and applications can be found in [1].

PSO algorithm

Pseudo code of the general PSO is given in Figure 1. In a PSO algorithm, population is initiated randomly with particles and evaluated to compute finesses together with finding the particle best (best value of each individual so far) and global best (best particle in the whole swarm). Initially, each individual with its dimensions and fitness value is assigned to its particle best. The best individual among particle best population, with its dimension and fitness value is, on the other hand, assigned to the global best. The best individual among particle best population, with its dimension and fitness value is, on the other hand, assigned to the global best. Then a loop starts to converge to an optimum solution. In the loop, particle and global bests are determined to update the velocity first. Then the current position of each particle is updated with the current velocity. Evaluation is again performed to compute the fitness of the particles in the swarm. This loop is terminated with a stopping criterion predetermined in advance.

Initialize parameters
Initialize population
Evaluate
Do{
    Find particlebest
    Find globalbest
    Update velocity
    Update position
    Evaluate
}While (Termination)

Figure 1. Simple PSO algorithm
The basic elements of PSO algorithm is summarized as follows:

- **Particle**: is a candidate solution \(i\) in swarm at iteration. The \(i^{th}\) particle of the swarm is represented by a \(d\)-dimensional vector and can be defined as \(X_i^k = \{x_{i1}^k, x_{i2}^k, ..., x_{id}^k\}\), where \(x\)'s are the optimized parameters and \(x_{id}^k\) is the position of the \(i^{th}\) particle with respect to \(d^{th}\) dimension. In other words, it is the value \(d^{th}\) optimized parameter in the \(i^{th}\) candidate solution.

- **Population**: \(pop^k\) is the set of \(n\) particles in the swarm at iteration \(k\), i.e., \(pop^k = \{X_1^k, X_2^k, ..., X_n^k\}\)

- **Particle velocity**: \(V_i^k\) is the velocity of particle \(i\) at iteration \(k\). It can be described as , \(V_i^k = \{v_{i1}^k, v_{i2}^k, ..., v_{id}^k\}\) where \(v_{id}^k\) is the velocity with respect to \(d^{th}\) dimension.

- **Particle best**: \(PB_i^k\) is the best value of the particle \(i\) obtained until iteration \(k\). The best position associated with the best fitness value of the particle \(i\) obtained so far is called particle best and defined as \(PB_i^k = \{pb_{i1}^k, pb_{i2}^k, ..., pb_{id}^k\}\) with the fitness function \(f(PB_i^k)\).

- **Global best**: \(GB^k\) is the best position among all particles in the swarm, which is achieved so far and can be expressed as \(GB^k = \{gb_{1}^k, gb_{2}^k, ..., gb_{d}^k\}\) with the fitness function \(f(GB^k)\).

- **Termination criterion**: it is a condition that the search process will be terminated. In this study, search is terminated when the number of iteration reaches a predetermined value, called maximum number of iteration.

3. Binary particle swarm optimization :

Kennedy and Eberhart proposed a discrete binary version of PSO for binary problems [23]. In their model a particle will decide on “yes” or “no”, “true” or “false”, ”include” or “not to include” etc. also this binary values can be a representation of a real value in binary search space. In the binary PSO, the particle’s personal best and global best is updated as in real-valued version. The major difference between binary PSO with real-valued version is that velocities of the particles are rather defined in terms of probabilities that a bit will change to one.

Using this definition a velocity must be restricted within the range \([0,1]\) . So a map is introduced to map all real valued numbers of velocity to the range \([0,1]\) [4]. The normalization function used here is a sigmoid function as:

\[ V_{ij}^k(t) = \frac{1}{1 + e^{-V_{ij}^k(t)}} \]

Also the equation (1) is used to update the velocity vector of the particle. And the new position of the particle is obtained using the equation below:

\[ x_{ij}^k(t+1) = \begin{cases} 1 & \text{if } r_{ij} < \text{sig}(V_{ij}^k(t+1)) \\ 0 & \text{otherwise} \end{cases} \]

Where is a uniform random number in the range \([0,1]\).

IV. RESULT & DISCUSSION

Using the methodology in Overture, it gives the following result shown in table 1 below:

<table>
<thead>
<tr>
<th>SCHEMA NAME</th>
<th>SYNTAX TYPE CHECK</th>
<th>DOMAIN CHECK</th>
<th>PROOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid (type)</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Prey (type)</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Predator (type)</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Perception (type)</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Toposense (function)</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Difference (function)</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Determine (function)</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Broadcast (function)</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Send (function)</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Receive (function)</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Table 1. Result of Formal Specification of BPSO algorithm

The screenshot of coding in Vienna Development method is shown in Fig.1

![Fig. 2 Screenshot of coding in VDM](image-url)
V. CONCLUSION & FUTURE WORK

Though Z, B and VDM are model based formal specification languages used for specifying user’s requirements in mathematical language that can be proved, verified and tested unambiguously. While the journey of all the three languages starts at the requirements specification phase of the software development life cycle (SDLC) model, but their path divides after this phase. Z works on high level abstraction of a system and provides a strong base for system designing and then testing it. However, B models the system in an abstract machine notation that can be used further to design system, generate its code and then refine and test the same. VDM is used to prove the equivalence of programming language concepts. They all do not differ radically from one another, but at some factors they differ a lot.

By seeing the different characteristics of all the three methods, it can be concluded that as Z and B has already been applied to compiler’s specification but still remain a challenge for error-free compiler. If this application is to be put through VDM, then this can be helpful for providing error-free compiler better than the previous work done.

All the operations of compiler are formally specified except the memory allocation and task scheduling. So memory allocation operation is done by formally specifying the binary particle swarm optimization algorithm. As Binary PSO has given a satisfiable result in formal specification, so it can be used in Task scheduling done by compiler also. In future, we will implement this concept so that the compiler can be proven error-free in all perspective.

REFERENCES
[10] Peter Gorm Larsen, “Recent Industrial Applications of VDM in Japan”.

