Journal of Software Engineering and Applications, 2012, 5, 873-882
http://dx.doi.org/10.4236/jsea.2012.511102 Published Online November 2012 (http://www.SciRP.org/journal/jsea)

Rapid and Flexible User-Defined Low-Level Hybridization for Metaheuristics Algorithm in Software Framework

S. Masrom*, Siti Z. Z. Abidin, N. Omar

Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Shah Alam, Malaysia.
Email: *suray078@perak.uitm.edu.my, sitizaleha533@salam.uitm.edu.my, nasiroh@tmsk.uitm.edu.my

Received September 23rd, 2012; revised October 21st, 2012; accepted October 30th, 2012

ABSTRACT

The metaheuristics algorithm is increasingly important in solving many kinds of real-life optimization problems but the implementation involves programming difficulties. As a result, many researchers have relied on software framework to accelerate the development life cycle. However, the available software frameworks were mostly designed for rapid development rather than flexible programming. Therefore, in order to extend software functions, this approach involves modifying software libraries which requires the programmers to have in-depth understanding about the internal working structure of software and the programming language. Besides, it has restricted programmers for implementing flexible user-defined low-level hybridization. This paper presents the concepts and formal definition of metaheuristics and its low-level hybridization. In addition, the weaknesses of current programming approaches supported by available software frameworks for metaheuristics are discussed. Responding to the deficiencies, this paper introduces a rapid and flexible software framework with scripting language environment. This approach is more flexible for programmers to create a variety of user-defined low-level hybridization rather than bounded with built-in metaheuristics strategy in software libraries.

Keywords: Software Framework; Scripting Language; Metaheuristics; Low-Level Hybridization; User-Defined Strategy

1. Introduction

Since the last decade, rapid software development has emerged as a preferable approach in software engineering especially for developing computer applications with complex computations such as distributed, scheduling and optimization systems [1]. The main critical factor for developing such kind of systems is time but complexity of programming development has led to very lengthy completion. As a result, in many aspects, rapid software development is able to provide easier, effective and productive method for each stage of the development. In order to facilitate rapid software development, a lot of innovations and improvements have been studied and invented. The ideas vary from diverse perspectives that include software design [2,3], software architecture [4], software modeling [5] and programming paradigm [6-8].

In the perspectives of programming paradigm, scripting language has been widely accepted by programmers for two reasons. Firstly, scripting language has simpler language structure as compared to other kinds of programming languages. Therefore, scripting language is usable for gluing together different complex algorithms from the low-level implementation of a software library [9]. Secondly, it can be used for developing strong functionality programs in a complex computer application such as distributed and collaborative application [10], grid computing [11] and agent based system [12].

In optimization based software, where the system foundation is structured with complex computational algorithm, scripting language programming is highly applicable for rapid development. Examples of complex computational algorithm for optimization application include metaheuristics and metaheuristics hybridization.

In many cases, metaheuristics hybridization is more useful for different kinds of real life problems [13]. The main intention for metaheuristics hybridization is to compensate one single metaheuristic limit with the strength of other algorithms.

The metaheuristics hybridization techniques can be classified either as high-level or low-level hybridization. The level of hybridization is used to differentiate the strength of combination between the hybrid algorithms [14,15]. In high-level hybridization, the components from different hybrid algorithms are not strongly dependent because the implementation retains original identity or behavior of the hybrid algorithms. In contrast,

*Corresponding author.

low-level hybridization requires internal structure modification, which committed with internal components exchange from the hybrid algorithms. Thus, the components from the hybrid algorithms are strongly connected to work together in finding optimal solutions.

The development of metaheuristics hybridization is very lengthy and difficult [5] especially to the low-level hybridization. Thus, in order to reduce the development time and difficulty, programmers might rely on software framework that provides software library for metaheuristics algorithm. Software framework promotes rapid implementation because the configurations for metaheuristics and hybridization in the software library could be simply specified with Graphical User Interface (GUI). GUI provides convenient and easy to learn interface for supporting interactions between software user (programmer) and software framework. However, in implementing low-level hybridization, GUI is not a suitable technique for enabling software library modification because the interactions between GUI functions and other functions is defined by built-in functions and application program interface (API) within the software library [16]. Therefore, in order to implement low-level hybridization with software framework, an easy and flexible programming environment at front-end software should be provided which could be supported by scripting language.

The remainder of this paper is organized as follows: Section 2 presents background of problem related to the software frameworks for metaheuristics. Then, the general concept of metaheuristics and formal definition of low-level hybridization is given in Section 3. The limitations of common programming approaches in software frameworks for metaheuristics are discussed in Section 4. Section 5 explains the proposed software framework before the concluding remarks in Section 6.

2. Background of the Problem

The research on rapid software framework for metaheuristics is still an ongoing research. Generally, there are two major issues in the development of rapid software framework for metaheuristics. These issues include implementation technique and algorithm strategy. **Figure 1** shows the related issues.

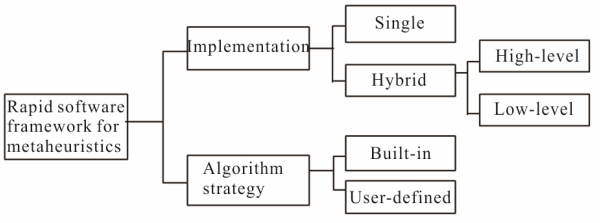

Figure 1. Development issues in rapid software framework for metaheuristics.

The first issue is categorized as metaheuristics implementation technique, which can be single or hybrid. Single implementation does not combine one metaheuristic paradigm with other algorithms but hybrid techniques involve cooperation of different algorithms. There are varieties of techniques applied metaheuristics hybridization. Generally, these techniques can be divided into high-level and low-level [14,15]. In high-level hybridization, the hybrid algorithms establish their communication through a well defined interface [14]. Therefore, the implementation does not involve internal structure modifications and retains original behavior of the hybrid algorithms. Besides, the components of hybrid algorithms are not dependent on each other. In contrast, low-level hybridization needs internal structure modification of the different hybrid algorithms. Thus, the original behavior of algorithms would be changed and different components from the hybrid algorithms are strongly connected to each other. Therefore, implementing low-level hybridization is more complicated than high-level hybridization.

Many researchers have agreed that relying on single metaheuristics is quite restricted in achieving best solution for real-life optimization problems [14,15]. Therefore, hybrid metaheuristics have been widely accepted as an effective approach as compared to single implementation. Nevertheless, many of available software frameworks facilitate more functionality on single metaheuristic [5]. This is due to the simpler structures of single metaheuristics. Some of the software frameworks for single implementation are JCLEC [17], JEO [18], TEA [19], JSwarm [20] and NetLogo [21]. *J*CLEC, JEO and TEA support Genetic Algorithm (GA) while NetLogo and JSwarm facilitate Particle Swarm Optimization (PSO) algorithm only.

However, a number of rapid software frameworks have been invented and introduced for hybrid metaheuristics such a HeuristicLab [22], ParadisEO [23] and Distributed BEAGLE [24]. Unfortunately, many of these software frameworks are majorly developed for high-level hybridization due to the fact that high-level hybridization is less complicated than low-level. Implementing high-level hybridization with software framework might involve simple configurations of combination that are not modifying internal structure of the hybrid algorithms. Therefore, the programmer can rely on pre-defined hybridization strategies located in software library.

Instead of the implementation techniques, algorithm strategy is the second issue in rapid software framework for metaheuristics. The algorithm strategy can be provided as built-in or user-defined [5]. Built-in strategies only supported pre-defined metaheuristics components in software library, which is less flexible and complex for

program amendment. In contrast, user-defined strategies allow programmers to extend or create new metaheuristics components according to their needs. With user-defined, a variety of new algorithm strategy such as hybridization scheme could be created at front-end software framework. As for example, with user-defined high-level hybridization, dynamic parallel algorithms could be implemented as provided in OpenTS software framework [25]. However, no low-level modification involves in implementing user-defined high-level hybridization. Hence the combination configurations could be defined simply with GUI or through template program.

Based on the reviewed literature, it can be concluded that software frameworks for metaheuristics and its hybridization are intentionally developed only to support rapid software development. The frameworks are not governed for flexible programming environment that support low-level hybridization as described in **Figure 2**. The grey box represents common features provided by the available software frameworks.

In addition, **Figure 2** also shows the limitation of the software frameworks in providing user-defined strategy for metaheuristics and its hybridization. User-defined strategy supports high flexibility for programmers to develop low-level metaheuristics hybridization. Many empirical experiments have shown that low-level hybridization is increasingly significant to metaheuristics strategy improvement [5]. However, difficulty in the development might discourage researchers for exploring new strategy.

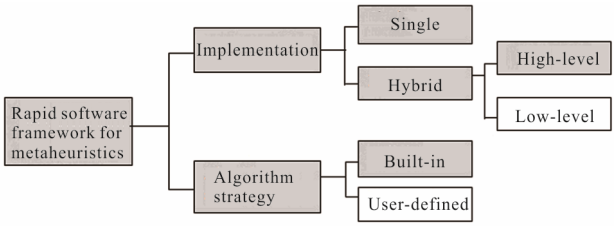

Figure 2. Common features of metaheuristics software framework.

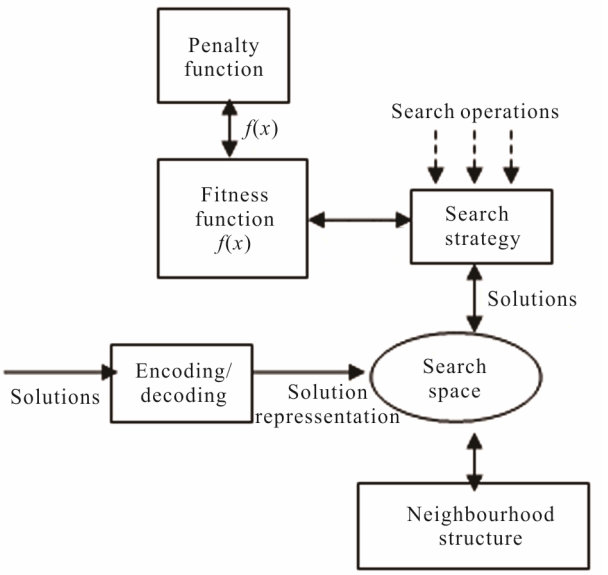

Figure 3. Metaheuristics components.

3. General Concept of Metaheuristics and Low-Level Hybridization

There are fives common questions for using metaheuristics in solving optimization problems. These questions are about the design of metaheuristics components which is composed of encoding method for solution representation, neighborhood structure, search strategy, fitness function and penalty function. **Figure 3** shows the relationship of each component.

The encoding and decoding are two essential steps in designing metaheuristics for a particular optimization problem. It must be suitable and relevant to the search operators and solution evaluation technique [26]. Therefore, the encoding and decoding plays an important role in the efficiency and effectiveness of a particular metaheuristics.

The search space consists of a finite set of decision variables for each solution representation. At the first search, the solutions will be randomly selected to go through fitness evaluation process according to a particular search strategy and objective. Different metaheuristics has different search strategy and operators.

Fitness function is an important element of a metaheuristics that formulates the search objective. It describes the quality or fitness of solutions and will guide the search toward better solutions [26]. Depending on the problem, a fitness function might be subjected to some kinds of constraint that is formulated in a penalty function.

The neighborhood structure is relatively important for some kind of metaheuristics. These metaheuristics are classified as single-based metaheuristics such as Simulated Annealing, Tabu Search and Variable Neighborhood Search. Different from population-based, the single-based metaheuristics implement generation and replacement of solutions from a single solution. The set of solutions for replacement is determined from the solution neighborhood. Therefore, in single-based metaheuristics, the structure of neighborhood plays a crucial role in their performance.

3.1. Metaheuristics Hybridization

Since every metaheuristics has distinct search strategy, each operator or component from different algorithms might be crossover to be new hybrid algorithms with

better performance. This process is called metaheuristics hybridization. When a metaheuristic changes its original paradigm by adding new components from other algorithms, it is described as implementing low-level hybridization [15].

In this paper, the metaheuristics that receive new components from other algorithms is referred to as master-metaheuristics. A master-metaheuristics can be included with new components from one or many sub-metaheuristics. Since the variation of low-level hybridization techniques is too broad, the hybridization techniques focus in this research is limited to hybridization among the family of population-based metaheuristics (p-metaheuristics) only. Besides, the modification process is restricted to be occurred in the search strategy component which is referred to as Proprietary components by [5]. The illustration of low-level hybridization for proprietary components of metaheuristics is given in **Figure 4**.

The details about low-level hybridization of p-metaheuristics are further described in the next sub-section as a formal definition.

3.2. Formal Definition of Low-Level Hybridization

The low-level hybridization of p-metaheuristics can be formally defined as a composition of (*m*,*s*) where:

- The $m, s \in M$ are different algorithms from the set of p-metaheuristics $M = \{a_1, a_2, \cdots, a_n\}$. The parameter *m* and *s* are devoted to master-metaheuristics and sub-metaheuristics respectively in which only one *m* can be integrated with more than one *s*.
- Each metaheuristic *M* is composed of general components $G = (S, f, \mathbf{\Omega})$ and proprietary components $C = \{x_1, x_2, \cdots, x_n\}$. While general components are common to all *M* algorithms, they have distinction with proprietary component.
- *S* is one of the general components that define solution representation in a search space. The search space consists of a finite set of decision variables $\mathbf{V_i}$ where $i = \{1, \cdots, n\}$. The type of variables can be in discrete, continuous or mixed form [27].
- Another general component for *M* is objective function $f = S \to R^+$ that assigns a cost value *MIN* and *MAX* to each solution of *S*. The set of constraints among the variables is defined in a set of penalty functions $\mathbf{\Omega} = \{f_1, f_2, \cdots, f_3\} \mid f_x : V_i \to C$ where $i = \{1, \cdots, n\}$.
- The proprietary components *C* are exclusive of their respective metaheuristics *M* which can be crossover into the routine of another metaheuristics *M*. It creates specific metaheuristics paradigm and characteristics that consists of different parameters with different types *T*. The parameters can be associated with dynamic or constant value. The dynamic values can be determined by dynamic behaviour either self-adaptive *A* or time-varying *V*. The sets of $A = \{a_1, a_2, \cdots, a_n\}$ and $V = \{v_1, v_2, \cdots, v_n\}$ consist of different functions with different types *T*. Each function calculates an appropriate parameter value in relation to a particular metaheuristics condition.

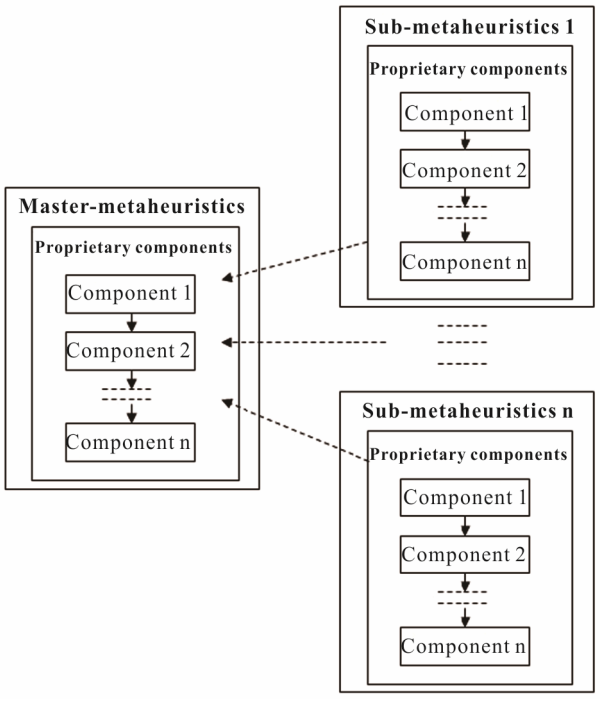

Figure 4. Low-level hybridization of proprietary components.

4. Programming Approach in Metaheuristics Software Framework

There are two types of programming approaches that can be implemented by programmers when developing metaheuristics with software framework as depicted in **Figure 5**.

The first type is through GUI approach at front-end software framework. This method is more suitable for simple operations such as to configure algorithms parameters or to call some executable functions but it has limited capability for implementing extensive program modification. Therefore, this approach has insufficient functions for supporting flexible user-defined low-level hybridization. In contrast, the second type of programming approach is related directly to software libraries which are located at back-end software framework. Usually, template program is provided for simple software operations but in order to perform extensive modification, the programmers have to explore and modify some of the software libraries. Although this approach is

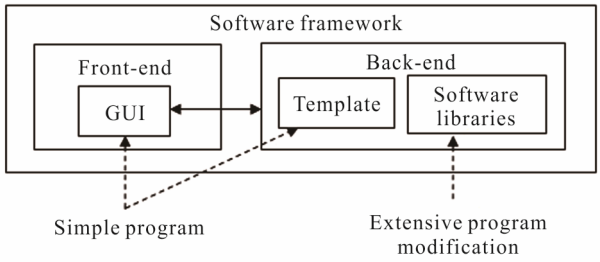

Figure 5. Programming approaches.

highly flexible than GUI, the task for modifying programs in software library is too complicated [5]. Since the process of finding the best hybridization strategy involves repeating tasks include redeveloping and validating, this approach is very costly and time consuming.

The implementation of flexible user-defined low-level hybridization is hindered by the limitation of these programming approaches. The GUI approach restricts users to modify software libraries while the template and software library modification involve tedious and complicated tasks. Nevertheless, these approaches are mostly adapted by the available software frameworks for metaheuristics. **Table 1** summarizes the available software frameworks with its programming approaches (GUI, template, software library) and implementation supports (single, high-level hybridization, low level hybridization).

As shown in **Table 1**, most of available software frameworks provide template programming approach and some software with GUI. Therefore, they are only capable to support single and high-level implementation due to the GUI and template limitations. Although available software frameworks are useful for low-level hybridization (e.g. MDF and ParadisEO), the strategies have been restricted to built-in metaheuristics components in software libraries. In order to perform extensive modification for user-defined strategy, the programmers have to modify software libraries which have been developed in JAVA, *C++* or *C#*. These programming languages are difficult for novice programmer.

Despite the fact that GUI is easier than template, it has restricted program modification. While hybridization techniques could be enabled in HeuristicLab with GUI, the implementation supports only for built-in high-level hybridization. In MDF software framework, the researchers introduce modeling technique for designing new hybridization strategies. The program codes are automatically generated based on user-defined model but the strategies are still bounded with available metaheuristics components in software library. In order to create new functions or components, the developer has to understand the detail of software working structure.

In order to resolve these deficiencies, an alternative solution is by providing scripting language programming environment at front-end software framework. While scripting language is less complicated to be used, it is also applicable for the development of complex functional programs. Therefore, scripting language can support rapid and flexible user-defined low-level hybridization for metaheuristics.

5. The Proposed Software Framework

The proposed software framework is designed with three-tier architecture namely front-end scripting language, intermediate compiler and back-end software libraries as illustrated in **Figure 6**.

Table 1. Available software frameworks for metaheuristics.

	GUI	Template	Single	Software library	High-level hybrid	Low-level hybrid
iOpt [28]	✓	x	✓	*JAVA*	**x**	x
Hotframe [29]	x	✓	✓	*JAVA*	✓	x
Mallba [30]	x	✓	✓	*JAVA*	✓	x
JEO [18]	x	✓	✓	*C++*	x	x
EasyLocal++ [31]	x	✓	✓	*C++*	✓	x
HeuristicLab [22]	✓	x	✓	*C#*	✓	x
JSwarm [20]	x	✓	✓	*JAVA*	x	x
MDF [5]	x	✓	✓	*C++*	✓	✓
TEA [19]	x	✓	✓	*C++*	x	x
OPT4J [32]	✓	✓	✓	*JAVA*	x	x
ParadisEO [23]	x	✓	✓	*C++*	✓	✓

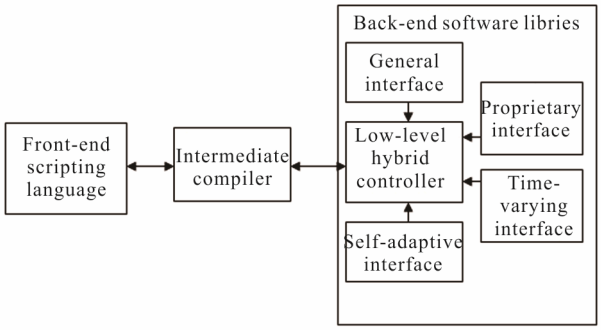

Figure 6. The proposed software framework.

5.1. The Back-End Software Library

The back-end software libraries consist of high performance program codes for different metaheuristics. High performance programs are developed with JAVA. The basic feature of the back-end software architecture is to support generic and extensible environment for implementing different kind of metaheuristics and optimization problems. The class architecture is illustrated in **Figure 7**.

The back-end programs are bundled with a collection of software libraries that consists of five main elements, one is a collection of controller classes and the rest is a set of interface classes. The set of interface classes comprise of general interface, proprietary interface, adaptive interface and time-varying interface. The interface classes are highly flexible for extensible through inheritance implementation. Therefore, the software frame-work can be used for developing different metaheuristics algorithms. In this paper, the focus is given to Particle Swarm Optimization (PSO) and Genetic Algorithm (GA).

The general interface consists of different classes for general metaheuristics components while proprietary interface provides specific component classes for specific metaheuristics. As shown in **Figure 7**, specific components for GA are selection, crossover and mutation while specific components for PSO are position and velocity update.

In addition, self-adaptive and time-varying interfaces facilitate dynamic behavior of metaheuristics parameters such as mutation rate for GA, inertia weight and constriction parameters for PSO. Self-adaptive behavior

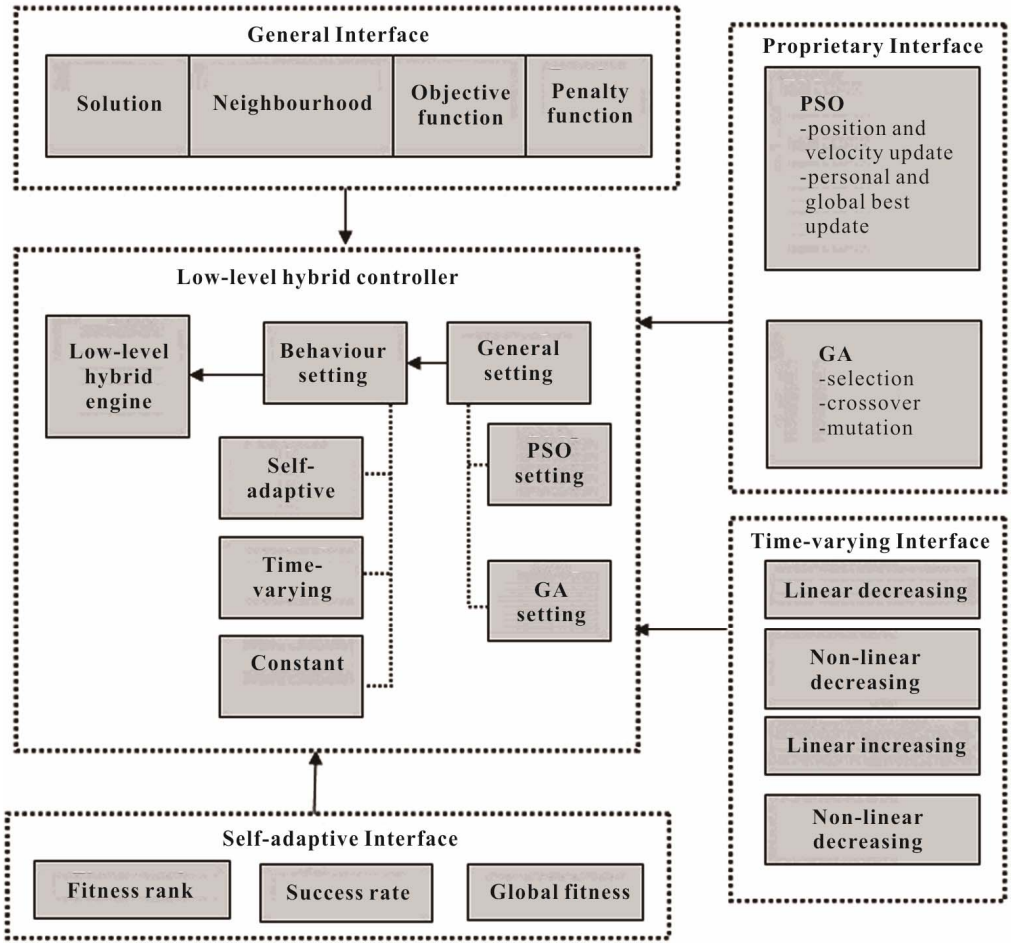

Figure 7. Back-end software architecture.

determines parameters value with regards to metaheuristics search ability namely exploration and exploitation. In metaheuristics, exploration ability promotes high accuracy of solutions while exploitation facilitates fast convergence speed. Therefore, the balance of both abilities is very important to the performances of metaheuristics algorithms. Based on empirical studies, it is found that one of the successful ways to balance the search ability is by providing self-adaptive environment [33]. Nevertheless, besides self-adaptive, time-varying behavior is also contributes to improvement of metaheuristics performance. By providing time-varying behavior, the exploration and exploitation search abilities can be dynamically adjusted according to number of iteration.

The low-level hybrid controller is an important element that consists of low-level hybrid engine for implementing low-level hybridization. It is a JAVA codes generated by scripting compiler at the intermediate software framework.

5.2. The Intermediate Compiler

Scripting compiler is intermediate software between scripting program and software library that translates the scripting language codes for low-level metaheuristics hybridization into JAVA program. The JAVA program will be saved at back-end software to operate as a main program that defines class interactions between software libraries. It might also consist of new instantiations of metaheuristics components.

5.3. The Front-End Scripting Language

The front-end software framework is supported with scripting language to be used by programmers for defining and developing new low-level hybridization of metaheuristics. The steps for basic operations in creating low-level hybridization are given in **Figure 8**.

The scripting language is not just usable for calling pre-defined operations from software libraries but capable in creating new functions of metaheuristics components. The scripting language is capable to fulfill these requirements since it has the following special characteristics:

- The scripting language tend not to have strong typing rules and not error-prone.
- In particular, scripting language tend to have powerful data structure and operations, which are tightly built-in with the language.

These special characteristics have given additional advantages to developers to use scripting language for many purposes such as to support rapid software development, API manipulation for back-end software libraries and software functions extension. These purposes are essential to the development of flexible user-defined low-level hybridization. The following parts discuss how scripting language can be used to fulfill these requirements.

1. Master metaheuristics declaration
2. Sub-metaheuristics declaration
3. Parameters declaration
4. Select behavior for parameters
   if(behavior is not available)→create new
5. Select the solution representation
   if(solution representation is not available)→create new
6. Select the neighbourhood structure
   if(neighbourhood structure not available)→create new
7. Select the problem function
   if(problem function not available)→create new
8. Create constraints for the problem
9. Select components from master-metaheuristics
   if(components not available)→create new
10. Select components from sub-metaheuristics
    if(components not available)→create new
11. Construct algorithm flows
    - must begin with solutions initialization
    - must consist of fitness function evaluation
    - must consist of master specific component
    - must consist of sub specific component

Figure 8. Basic operations of low-level hybridization.

5.4. Scripting Language for Rapid Software Development

Scripting languages are simpler than application programming languages like C, C++ and JAVA. Thus, scripting language will considerably increase the efficiency of development. Some of the key enhancing features in scripting languages with regards to efficiency include simplify, expressiveness, and easiness.

Scripting language consists of simplify instructions which reduces the development errors for programming. A single statement of a scripting language can describe many instructions. In other words, scripting languages are normally expressiveness. It can generate roughly 10 to 100 times shorter code compared to application programming languages. For example, one statement in Tcl can represents more than 100 instructions while in C programming language, it means for several (5 to 10) instructions only.

5.5. Scripting Language for API

Scripting language can be used to call and communicate with different program codes from the software libraries [34]. Thus, scripting language can be used to operate as an API between the different programs or applications software.

The API with scripting language could be implemented at front-end software framework while the software libraries at back-end software are written with ap-

plication programming language such as JAVA or C++. Therefore, scripting language and application programming language are complimentary to each other. Programming with scripting language promises better productivity for integrating different software codes and libraries while application programming language supports high efficiency performance for algorithm implementation.

5.6. Scripting Language for Flexible Software Extension

At the beginning, conventional scripting languages have some limitations, which restrict their efficiency and expressiveness. For instance, they have limited capability for concurrency, data structuring and object-oriented programming. Responding to the deficiencies, language developers have created advanced scripting languages such as Perl, Python and among others. Perl provides compact but very powerful sets of data types and data structures for example list, stack and queue. Python also has a great feature comparable Perl, but it is based on object-oriented programming. In addition, the Tcl is an excellent language for writing system or hardware program without using intermediate language to machine. With the very powerful sets of data types and data structures, scripting languages are not just able to invoke and glue together other programs, but they are also appropriate for writing new complex program as flexible as application programming language.

With the flexibility, software designer interest has been attracted to invent different types of scripting languages for different types of computer applications. As for example, the JACIE scripting language has been invented for developing collaborative and distributed application [10]. In addition, in order to develop grid based application Gaussion script has been introduced [11]. More than that, a group of researcher has created Japlo scripting language useful for rules based programming similar to Prolog language [35]. The Japlo language works on top of JAVA and it has better functionality than the Prolog itself.

In addition, there are also available scripting languages for implementing metaheuristics based application. **Table 2** lists the scripting languages for metaheuristics.

Some scripting use XML script which is integrated in metaheuristics software framework such as Open Beagle [36], Distributed Beagle [24] and JCLEC [17]. The XML in JCLEC is used for running different experiments with different configurations in parallel.

A scripting programming language is proven to be useful for facilitating user-defined strategy. The T++ scripting language for example, has been designed to support flexible user-defined strategy [25]. The language is incorporated in OpenTS software framework but it is only usable for high-level hybridization. More interesting, in METASIS software framework [37], the scripting with Sequential Interactive Synthesis System (SIS) could enable user-defined strategy for low-level hybridization. However, SIS is specifically usable for synthesizing and optimizing sequential circuits only. Thus, it has a limit to be employed for different kinds of optimization problem.

6. Conclusions

Rapid programming is significant to the development of metaheuristics based applications. Therefore, the implementation needs software framework in order to reduce development time and difficulty. However, for implementing low-level metaheuristics hybridization with software framework, another complexity has emerged due to its extensive programming. Current approaches in software frameworks have some limitations for supporting easy and flexible programming. As a result, they lack of flexibility in providing user-defined hybridization strategy at front-end software. Besides, the deficiencies would discourage programmers to develop a variety of low-level hybridization strategies for metaheuristics algorithms.

In order to enable rapid and flexible front-end programming environment, a scripting language should be developed. The scripting language is not just beneficial for integrating different metaheuristics programs, but it also permits the creation of new functionalities beyond the capability of software libraries. This paper has several contributions to the metaheuristics community as well as to the software engineering. Firstly, formal definition of low-level hybridization provides general views of the implementation. Secondly, it introduces three-tier software architecture that promotes rapid and flexible

Table 2. Scripting language for metaheuristics.

Scripting language	Software library	User-defined strategy	High-level hybrid	Low-level hybrid
XML [36]	C++	✓	x	x
XML [24]	C++	x	✓	x
T++ [25]	C++	✓	✓	x
SIS [37]	SIS	✓	x	✓

programming at front-end software. In addition, this paper also provides operation steps for operating low-level metaheuristics hybridization that can be used for designing structure and details specification of a software framework.

7. Acknowledgements

The authors would like to thank Kementerian Pengajian Tinggi MALAYSIA and Universiti Teknologi MARA for their financial support to this project.

REFERENCES

[1] B. Boehm, "A View of 20th and 21st Century Software Engineering," *Proceedings of the 28th International Conference on Software Engineering*, Shanghai, 2006, pp. 12-29.

[2] E. Dubois, P. Gray and L. Nigay, "ASUR++: Supporting the Design of Mobile Mixed Systems," *Interacting with Computers*, Vol. 15, No. 4, 2003, pp. 497-520. doi:10.1016/S0953-5438(03)00037-7

[3] A. Neyem, S. F. Ochoa, J. A. Pino and R. D. Franco, "A Reusable Structural Design for Mobile Collaborative Applications," *Journal of Systems and Software*, Vol. 85, No. 3, 2012, pp. 511-524.

[4] R. Weinreich and G. Buchgeher, "Towards Supporting the Software Architecture Life Cycle," *Journal of Systems and Software*, Vol. 85, No. 3, 2012, pp. 546-561. doi:10.1016/j.jss.2011.05.036

[5] H. C. Lau, W. C. Wan, S. Halim and K. Toh, "A Software Framework for Fast Prototyping of Meta-Heuristics Hybridization," *International Transactions in Operational Research*, Vol. 14, No. 2, 2007, pp. 123-141. doi:10.1111/j.1475-3995.2007.00578.x

[6] S. H. Sadat-Mohtasham and A. Ghorbani, "A Language for High-Level Description of Adaptive Web Systems," *Journal of Systems and Software*, Vol. 81, No. 7, 2008, pp. 1196-1217. doi:10.1016/j.jss.2007.08.033

[7] S. Z. Z. Abidin, M. Chen and P. W. Grant, "Designing Interaction Protocols Using Noughts and Crosses Type Games," *Journal of Network and Computer Applications*, Vol. 30, No. 2, 2007, pp. 586-613. doi:10.1016/j.jnca.2006.01.002

[8] T. C. Oliveira, P. S. C. Alencar, C. J. P. de Lucena and D. D. Cowan, "RDL: A Language for Framework Instantiation Representation," *Journal of Systems and Software*, Vol. 80, No. 11, 2007, pp. 1902-1929. doi:10.1016/j.jss.2007.01.005

[9] A. Boulis, C.-C. Han and M. B. Srivastava, "Design and Implementation of a Framework for Efficient and Programmable Sensor Networks," *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, 2003, pp. 187-200.

[10] M. A. Ismail, M. Chen and P. Grant, "JACIE—An Authoring Language for WWW-Based Collaborative Applications," *Annals of Software Engineering*, Vol. 12, No. 1, 2001, pp. 47-75. doi:10.1023/A:1013354519370

[11] W. Tongming, Z. Ruisheng, S. Xianrong, C. Shilin and L. Lian, "GaussianScriptEditor: An Editor for Gaussian Scripting Language for Grid Environment," *Proceedings of the 8th International Conference on Grid and Cooperative Computing*, 2009, pp. 39-44.

[12] X. Hua, L. Qingshan, W. Yingqiang, Z. Chenguang, M. Shaojie and Z. Guilin, "A Scripting Language Used for Defining the Integration Rule in Agent System," *IEEE International Conference on E-Business Engineering*, 2008, pp. 649-654.

[13] X. Fu, A. Li, L. Wang and C. Ji, "Short-Term Scheduling of Cascade Reservoirs Using an Immune Algorithm-Based Particle Swarm Optimization," *Computers & Mathematics with Applications*, Vol. 62, No. 6, 2011, pp. 2463-2471. doi:10.1016/j.camwa.2011.07.032

[14] C. Blum and A. Roli, "Hybrid Metaheuristics: An Introduction," In: C. Blum, M. J. B. Aguilera, A. Roli and M. Sampels, Eds., *Hybrid Metaheuristics*, Springer, Berlin/ Heidelberg, 2008, pp. 1-30. doi:10.1007/978-3-540-78295-7_1

[15] E. G. Talbi, "A Taxonomy of Hybrid Metaheuristics," *Journal of Heuristics*, Vol. 8, No. 5, 2002, pp. 541-564. doi:10.1023/A:1016540724870

[16] D. Orenstein, "Application Programming Interface (API)," *Quick Study*: *Application Programming Interface* (*API*), 2000.

[17] S. Ventura, C. Romero, A. Zafra, J. A. Delgado and C. Hervás, "JCLEC: A Java Framework for Evolutionary Computation," Springer, Berlin/Heidelberg, 2008.

[18] M. G. Arenas, N. Dolin, J. J. Marelo, P. A. Castillo, I. F. de Viana and M. Schonauer, "JEO: JAVA Evolving Objects," *Proceedings of the Genetic and Evolutionary Computation Conference* (*GECCO*), 2002.

[19] M. Emmerich and R. Hosenberg, "TEA : A C++ Library for the Design of Evolutionary Algorithms," University of Dortmund, Dortmund, 2001.

[20] C. Pabl, "JSwarm-PSO," 2006. http://jswarm-pso.sourceforge.net/

[21] F. Stonedahl and U. Wilensky, "NetLogo Particle Swarm Optimization Model." http://ccl.northwestern.edu/netlogo/models/

[22] S. Wagner and M. Affenzeller, "HeuristicLab: A Generic and Extensible Optimization Environment," In: B. Ribeiro, R. F. Albrecht, A. Dobnikar, D. W. Pearson and N. C. Steele, Eds., *Adaptive and Natural Computing Algorithms*, Springer, Vienna, 2005, pp. 538-541. doi:10.1007/3-211-27389-1_130

[23] S. Cahon, N. Melab and E. Talbi, "ParadisEO : A Framework for the Reusable Design of Parallel and Distributed Metaheuristics," *Journal of Heuristics—Special Issue on New Advances on Parallel Meta-Heuristics for Complex Problems*, Vol. 10, No. 3, 2004, pp. 357-380.

[24] M. Dubreuil and M. Parizeau, "Distributed BEAGLE : An Environment for Parallel and Distributed Evolutionary Computations," *Proceedings of the 17th Annual International Symposum on High Performance Computing Systems and Applications*, 2003.

[25] S. Abramov, A. Adamovich, A. Moskovsky, E. Shevchuk,

Y. Shevchuk and A. Vodomerov, "OpenTS: An Outline of Dynamic Parallelization Approach," In: V. Malyshkin, Ed., *Parallel Computing Technologies*, Springer-Verlag, Berlin/Heidelberg, 2005, pp. 303-312. doi:10.1007/11535294_26

[26] E. G. Talbi, "Metaheuristics: From Design to Implementation," Wiley, London, 2009.

[27] G. R. Raidl, J. Puchinger and C. Blum, "Metaheuristic Hybrids," In: M. Pardalos, H. Panos, P. Van and M. Milano, Eds., *Handbook of Metaheuristics*, Springer, New York, 2010.

[28] L. Rapha and C. Voudouris, "HSF: The iOpt's Framework to Easily Design Metaheuristic Methods," In: M. G. C. Resende and J. P. de Sousa, Eds., *Metaheuristics*: *Computer Decision-Making*, Kluwer Academic Publishers, Berlin/Heidelberg, 2004, pp. 237-256.

[29] A. Fink and S. Voß, "Hotframe: A Heuristic Optimization Framework," In: S. Voß and D. L. Woodruff, Eds., *Optimization Software Class Libraries*, Springer, Heidelberg, 2002, pp. 81-154.

[30] E. Alba, *et al*., "MALLBA: A Library of Skeletons for Combinatorial Optimisation," In: R. Feldmann, Ed., *Euro-Par* 2002 *Parallel Processing*, Springer-Verlag, Berlin/ Heidelberg, 2002, pp. 63-72. doi:10.1007/3-540-45706-2_132

[31] L. D. Gaspero and A. Schaerf, "EASYLOCAL++: An Object-Oriented Framework for Flexible Design of Local Search Algorithms," *Software-Practice and Experience*, 2003, pp. 1-34.

[32] M. Lukasiewycz, M. Glaß, F. Reimann and D.-I. Sabine Helwig, "The OPT4J Documentation," 2009.

[33] K. Suresh, S. Ghosh, D. Kundu, A. Sen, S. Das and A. Abraham, "Inertia-Adaptive Particle Swarm Optimizer for Improved Global Search," *Proceedings of the* 8*ht International Conference on Intelligent System Design and Applications—ISDA*2008, Kaohsiung, 26-28 November 2008, pp. 253-258.

[34] G. Zao-Bin, L. Ching and V. Varadharajan, "A Middle-Ware-Based Script Language," *Proceedings of the International Conference on Mobile Business* (*ICMB*'05), 11-13 July 2005, pp. 690-693. doi:10.1109/ICMB.2005.8

[35] M. Espak, "Japlo : Rule-Based Programming on Java," *Journal of Universal Computer Science*, Vol. 12, No. 9, 2006, pp. 1177-1189.

[36] C. Gagn and M. Parizeau, "Open BEAGLE : A New Versatile C++ Framework for Evolutionary Computations," *Proceedings of the Genetic and Evolutionary Computation Conference* (*GECCO*-2002), 2002.

[37] G. Ranjan, P. Kumar and P. Gupta, "METASIS: A Meta Heuristic Based Logic Optimizer," *Proceedings of the* 50*th Midwest Symposium on Circuits and Systems*, Montreal, 5-8 August 2007, pp. 1501-1504. doi:10.1109/MWSCAS.2007.4488825

