Journal of Network and Innovative Computing
ISSN 2160-2174 Volume 3 (2016) pp. 192-199

Improving UML Class Attribute Definitions Using Particle Swarm Optimization

Renu George[1], Philip Samuel[2]

[1] College of Engineering, Chengannur, Kerala, India
renugeorge@ceconline.edu

[2] Information Technology, SOE, Cochin University of Science and Technology, Kochi, Kerala, India
philips@cusat.ac.in

***Abstract*: Unified Modeling Language has become the de-facto industry standard for object-oriented modeling of the static and dynamic aspects of software systems. Class diagrams represent the static aspects of the system, the classes required for implementation of the system, the relationships between classes and the attributes and methods of each class. Attributes describe the data contained in an object of a class and its properties such as name, data type, visibility etc. Methods define the ways in which objects interact. Activity diagram represents the dynamic behavior of the system. The implementation of methods in a class is depicted using activity diagram. To ensure software quality, it is essential to maintain consistency between diagrams of the same model. Class diagrams can be mapped directly to an object oriented programming language and inconsistency in attribute definitions may be reflected directly in the generated code. Complex systems require large number of diagrams and hence detection of inconsistencies in class attribute definitions has a significant role during the design phase of software development. In this paper we describe a method for improving the class attribute definitions using particle swarm optimization technique. Particle Swarm Optimization (PSO) is a soft computing technique that provides solutions to optimization problems by maximizing certain objectives in a complex search space. The PSO algorithm is applied to detect inconsistency in attribute definitions and to optimize the fitness value of the attributes. The application of PSO algorithm improves the attribute definitions and provides consistent, optimized diagrams that result in the generation of more accurate code.**

***Keywords*:** class diagram, activity diagram, class attribute, consistency, particle swarm optimization

I. Introduction

Unified modeling language (UML) has become the de-facto standard for requirements modeling and is the most popularly used modeling language. UML provides a set of diagrams to model the requirements. A single diagram is not capable of representing all the aspects of the system and different models are built to capture the static and dynamic aspects of the system [39]. The elements of the system and their relationships are described using static model. Class diagrams represent the static aspects of the system, the classes required for implementation of the system, the relationships between classes and the attributes and methods of each class. The behavior of the system over time is represented using dynamic models. UML also provides different diagrams such as activity diagram, sequence diagram, state chart diagram and collaboration diagram to model the dynamic aspects of the system.

Large complex systems involve different models. The model may be composed of several diagrams and consistency should be maintained among the diagrams of the same model. The whole set of diagrams should not contradict one another since they are descriptions of the aspects of a single system [39]. The refinement of the models may also introduce inconsistencies. Inconsistency can be referred to conflicting requirements, violation of constraints and any design model possessing these properties is termed inconsistent. There are two types of consistency issues: intra-model consistency and inter-model consistency issues [23]. Consistency expected between different model elements of the same version depicting the static and dynamic aspects is termed intra-model consistency or horizontal consistency and between different versions of the same model is termed inter-model consistency or vertical consistency. Identification of inconsistencies during the design phase results in the development of accurate code. Manually detecting the inconsistencies is tedious, time consuming and prone to errors.

A recent and emerging paradigm in bio-inspired computing for implementing adaptive systems is Swarm Intelligence (SI). The behavior of real world insect swarms is considered as the basis of Swarm Intelligence and the behavior is used as a problem solving tool in SI. A population of interactive elements that performs a collaborative search of space to optimize some global objective is called swarm [18]. The original intent of particle swarm concept was to simulate the choreography of a bird flock or fish school and it originated as a simulation of simplified social system. However, it was found that particle swarm model can be used as an optimizer [5].

Particle swarm optimization (PSO) proposed by Kennedy and Eberhart in 1995 is inspired by the social behavior of bird flocking or fish schooling searching for food. PSO is a computational intelligence oriented, stochastic, population-based global optimization technique [2]. In terms of memory requirement and speed, it is computationally inexpensive and requires only primitive operators. PSO has a

simple concept with a unique searching mechanism, computational efficiency, and easy implementation and has been extensively applied to many engineering optimization areas.

The process of finding the maximum or minimum value of a function or process satisfying a set of constraints is termed optimization. The goal of optimization is to maximize efficiency. Consistency checking can be treated as a form of optimization that checks each component of the UML class and activity diagram, verifies whether the attributes involved in the implementation of the activity is consistent and optimizes the fitness value of attributes [3]. The attributes in a class diagram has a set of properties that fully describe the attribute such as the class in which it is defined, name, type, visibility etc. The fitness value is defined as a function of the properties of the attributes. We have defined two types of inconsistency related to attribute specification: attribute missing inconsistency and attribute specification inconsistency. In this paper, we focus on improving the UML class attribute definition by using PSO algorithm. Intra-model consistency problems in attribute definitions between class diagram and activity diagrams are identified by applying the principle of PSO. The PSO algorithm analyzes the consistency value of attributes to detect and fix attribute missing and attribute specification inconsistency. Manually detecting and fixing inconsistencies in complex systems involving large number of models is time consuming and error prone. Identifying and fixing the inconsistencies in the design phase of software development results in the generation of accurate software.

II. Related Work

Detecting inconsistency in attribute definition in UML diagrams is very crucial to the development of accurate software. The paper rule based detection of inconsistency in UML models [4] defines a production system language and rules specific to software designs modeled using UML class diagram and sequence diagram. Inconsistencies can be detected, users can be notified, solutions can be recommended and inconsistencies can be automatically fixed using this approach.

Inconsistency between different versions of a UML model expressed as a collection of class, sequence and state diagrams is detected and resolved in [6] using description logic. UML model is specified as a collection of concepts and roles using a knowledge representation tool, Loom. Detection and resolution of inconsistencies are performed using logic rules.

A metamodel independent method for checking model consistency is proposed in [8]. Here models are represented by sequence of elementary construction operations and consistency rules are then expressed as logical constraints on such sequences. The approach mainly deals with class diagram and use case diagrams.

A method on specifying consistency rules on different aspect models expressed in UML is specified in [9]. Consistency checking and consistency rules are also proposed. The paper instant consistency checking for UML [10] presents an approach for dealing with model changes and quickly, correctly, and automatically deciding what consistency rules to evaluate. A method of checking inconsistencies in UML models using description logic is specified in [11].

The application of swarm optimization technique in software development environment is specified in [12]. An algorithm to provide decision making support for class responsibility assignment in a class diagram by reassigning methods and attributes to classes using Particle Swarm Optimization is presented in the paper. The building blocks of swarm intelligence and how they are used to solve the routing problem is discussed in [13]. A general framework called Ant Colony Routing in which most swarm intelligence routing algorithms can be placed is presented in the paper.

A possible solution for the optimal operation of distribution networks which takes into account the impact of Distributed Generations is discussed in [14]. The distribution problem planning is done using PSO. Detection of concurrency problems such as deadlock and starvation using Particle Swarm Optimization algorithm is proposed in [15]. Thread execution interleaving that has a high probability of revealing deadlock and starvation faults are optimized by PSO which results in reduced complexity and increased accuracy.

A framework to test the consistency of data coming from different types of data sources is described in [16]. Consistency requirements are modeled using Object Constraint Language (OCL) and other modeling elements and model instances should be represented in XMI format for consistency check. An analysis of the PSO algorithm is performed in [20]. PSO is a simple algorithm with fewer parameters.

Consistency issues in behavioral models are dealt with in [17] and a methodology for dealing with consistency problems is presented. Consistency tests are formulated by mapping the relevant aspects of the model to a semantic domain and the methodology is applied to concurrent models in UML-RT.

An adaptive PSO algorithm to perform automatic tracking of the changes in a dynamic system by adding two new changes to PSO: environment detection and response is described in [21]. Functional test case selection based on multi-objective PSO is discussed in [22]. Selection of functional test cases considering coverage criteria and requirements effort using multi-objective PSO is investigated in the paper.

The consistency problems arising between class diagram (static diagram) and state diagram (dynamic diagram) is studied in paper [28]. Consistency semantics for class and state machines consisting of an Object-Z class describing the data aspects of a class and an associated state machine is studied. Consistency definition to the whole specification is provided by means of a translation to a common semantic domain. A technique for automatically checking each definition with a model checker is provided.

A framework based on OCL to check the consistency of UML models is described in [29]. The consistent models can be transferred from the checker tool to any other UML tool using the XMI standard. The paper highlights some shortcomings in the UML definition and also proves the support provided by OCL in managing tool peculiarities.

A methodology that tackles impact analysis and change management of analysis/design documents in the context of UML-based development is presented with the support of a prototype tool (iACMTool) [30]. OCL constraints on an adaptation of the UML meta-model are used to formally

define impact analysis rules, consistency rules between UML diagrams, automated change identification and classification between two versions of a UML model.

An approach using Boolean satisfiability to solve UML/OCL verification problems is discussed in [31]. The encoding and solution of the respective problem components of a verification problem, namely system states of a UML model, OCL constraints, and the actual verification task using an off the-shelf SAT solver is discussed.

A formal (mathematical) definition of the UML class diagram and its semantics is provided in [32]. A precise definition of the particular elements of the diagram and an interpretation of the whole diagram is also provided. The paper provides a reasoning of the class diagram and the inconsistencies that can occur in the diagram.

The various model consistency conceptions, proposals, problems and solutions are identified through a systematic literature review (SLR) in [33]. The review resulted in the identification of open issues, trends and future research within this scope of UML model consistency management. A preliminary formal approach to solve consistency problems based on transformation languages and rewriting logic is also provided.

A formal tool support to address the problem of inconsistency management in evolving UML models is discussed in [34]. A classification of semantic inconsistencies in and between evolving class diagrams, sequence diagrams, and state diagrams is provided followed by a UML profile to support versioning and evolution. Logic rules based on description logic was used to provide tool support. The UML models in XMI format was translated into the logic database of the description logics tool using an XML translator.

The issue of consistency of behavioural models in UML and techniques for specifying and analyzing consistency is discussed in paper [35]. The elements of UML model are transformed into a semantic domain using meta-model rules and consistency constraints specified and validated using the language and tools of the semantic domain.

Automated support in fixing inconsistencies in UML models is provided in [36]. Inconsistencies are fixed by automatically generating a set of concrete changes for an inconsistency and information about the impact of each change is provided on all consistency rules. A rigorous analysis technique for UML is provided in [37]. The analysis is based on the use of diagrammatical transformations. A formal semantics for a small subset of the language of UML class diagrams is developed. Deductive transformation rules are developed based on these semantics.

A study on the tractable consistency checking of UML diagrams is proposed in [41]. Inconsistencies in the diagrams are identified by translating them into first-order predicate logic that is generalized by counting quantifiers. A restricted set of class diagrams are considered which is produced by deleting certain components. The approach generated optimized algorithms for testing consistencies of restricted class diagrams

Class diagrams are an important part of any UML model because all later design and implementation work is based on this foundation. Good quality class diagrams result in the development of high quality software. A survey of the existing class diagram metrics is provided in [38]. Measuring the quality of class diagrams helps object oriented software designers to identify weak design spots when the cost of improvement is less, to choose between design alternatives, predict external quality attributes such as maintainability and reusability and improve resource allocation based on these predictions.

The application of PSO in dynamic environment is proposed in [42]. The paper presents two PSO techniques for an efficient and robust optimization over the dynamic systems, namely Fractional Global Best Formation (FGBF) with multi-swarms and Multi-Dimensional (MD) PSO. The particle structure and the swarm guidance are upgraded using the proposed techniques with substantial improvements in terms of speed and accuracy. Software design involves a set of models and each model in the multimodel is termed a partial model. A framework that specifies overlap between models as a network of inter-related metamodels or metamodel schema and defines consistency is specified in [43].

A mixed scheduling algorithm which is a combination of PSO and simulated annealing is proposed in [44]. The improved particle swarm optimization algorithm applied to a cloud environment reduces the operation time of tasks, provides efficient and proper scheduling of resources to a task and increases resource utilization ratio.

III. Particle Swarm Optimization

PSO is a machine learning technique based on the collective movement of a number of particles or birds in search of global optimum. The potential solutions are called particles and each particle has a position, velocity and fitness value. The fitness function computes the fitness value of the particles. The particles move by following the current optimum particle in the search space and the movement is directed by velocity. Each particle communicates with its neighbors about its performance, records its best performance so far and knows the position of the highest performing neighbor. The position of each particle is updated based on the displacement at the previous time step in the same direction it was following; the displacement in the direction towards the position where the highest performance of the particle so far was recorded; and displacement in the direction towards the position of the highest performing neighbor at that moment [1]. At every iteration, the position of the particle relative to the goal is evaluated and the best position of the particles in the neighborhood is shared with this particle and this information is used by the particle to update its position and velocity [19].

The search space is initialized with a group of random particles (solutions) and search for optima is performed by updating generations. The particles are updated by following two best values in every iteration: pbest and gbest. The best solution or fitness achieved by the particle so far is called pbest or personal best and gbest or global best represents the best value obtained so far by any particle in the population [5]. The particle updates its velocity and positions after finding the two best values, with the equations (1) and (2).

$$v[t+1] = w * v[t] + c1 * rand(0,1) * (pbest - present[t]) + c2 * rand(0,1) * (gbest - present[t]) \quad (1)$$

$$present[t+1] = present[t] + v[t+1] \quad (2)$$

where
v[t] is the velocity of the particle at time t,
w is the inertia weight
present[t] is the current particle (solution) at time t,
pbest is the best value attained by the particle so far,
gbest is the best value attained so far by any particle in the population,
rand() is a random number between 0 and 1,
c1, c2 are learning factors.

The pseudo code for PSO algorithm [5] is as follows:

```
Initialize particles and parameters of PSO
Do
  For each particle
    Calculate fitness value
    If the fitness value is better than the best fitness value
      set current value as the new pbest
  End
  Choose the particle with the best fitness value of all the particles as the gbest
  For each particle
    Calculate particle velocity according equation (1)
    Update particle position according equation (2)
  End
While maximum iterations or minimum error criteria is not attained
```

The PSO algorithm consists of an initialization part that initializes the particles in the search space, computation of fitness value and updating the velocity and position of the particles. The steps are iterated until minimum error criteria or the maximum number of iterations is reached.

A. Parameters of PSO

The key steps in applying the PSO are the representation of the solution and the fitness function. One of the advantages of PSO is that real numbers can be taken as particles and a standard procedure can be applied to find optima. The searching process is applied repetitively until optima is reached or the maximum number of iterations is encountered. The PSO algorithm has a set of parameters that are to be initialized [5].

1) Number of particles

The typical range is 20 to 40. For most of the problems good results can be obtained with 10 particles.

2) Dimension of the particles

The dimension of the particles depends on the problem to be optimized.

3) Range of particles

Range of particles depends on the problem to be optimized and different ranges can be specified for different dimensions of particles.

4) Vmax

Determines the maximum change of a particle during one iteration. It improves the resolution of search by limiting the velocity of particles. The value is set by the programmer.

5) Learning factors

The learning factors c1 and c2 are called the cognitive and social scaling parameters. The cognitive component, c1 acts as the particle's memory and causes the particles to return to the regions of the search space in which it has experienced high individual fitness. The social component c2, represents the size of the step the particle takes toward gbest and causes the particle to move to the best region the swarm has found so far. Usually c1 equals to c2 and ranges from [0, 4].

6) Stopping criteria

The stopping criteria represent the minimum error condition or the maximum number of iterations the PSO algorithm executes. The stopping condition also depends on the problem to be optimized.

7) Inertia weight

The concept of inertia weight is introduced by Shi and Eberhart. The contribution rate of a particle's previous velocity to its velocity at the current time step is determined by the inertia weight. Global search is facilitated by large inertia weight and local search by small inertia weight [24]. The movement of the particle in the same direction it was heading is determined by the inertia component [25]. The value of the inertia coefficient w is typically between 0.8 and 1.2 [26].

IV. Consistency Checking of Class Attributes

Consistency checking consists of analyzing the models to identify unwanted configurations and the model is inconsistent if such configurations are found [40]. The system is designed using UML class and activity diagrams and the entire model is analyzed to detect model inconsistency. Class diagrams provide information about the attributes and methods in a class. The values that can be attached to instances of class or interface are defined using attributes. An attribute is defined in terms of its properties such as name, data type, visibility, multiplicity etc. The visibility of a method, its return type and the parameters of the method can also be specified in the class. Object oriented programming languages are based on the concept of classes and hence class diagrams can be directly mapped with object oriented languages. Due to this property class diagrams are widely used at the time of construction. The static view of the system can be visualized using class diagrams and they can also be used to construct the executable code for forward and reverse engineering of any system.

Activity diagrams depict the dynamic behavior. The implementation of methods in a class diagram is specified using an activity diagram. Activity diagrams represent how activities are co-ordinated to implement an operation. The essential dependencies between different activities and the order in which they are to be performed is depicted using activity diagram [39]. An activity diagram is a collection of nodes and edges. Edges depict control flow. Nodes represent steps in the implementation of an operation.

The attributes can be used to construct executable code and hence consistency checking of attribute definitions has a significant impact on the accuracy of the code generated. The nodes of the activity diagram represent the actions performed to implement a method. Our method of consistency checking performs an analysis of the activity diagram nodes, identifies the attributes involved and verifies whether the attributes are defined in the class and is consistent with respect to its specification of properties. The aim of our work is to detect inconsistencies in attribute definitions and optimize the attribute specification.

The attributes are assigned a fitness value. The fitness value determines the consistency of the attributes. Inconsistencies

are detected by analyzing the fitness value of attributes. We have identified two types of inconsistencies: a) Attribute missing inconsistency and b) Attribute specification inconsistency.

A. Attribute Missing Inconsistency

An activity diagram represents an operation or function performed by the system and provides a graphical representation of the steps in the implementation of the function. The nodes specify the actions to be performed in implementing the function. Implementation of an action involves attributes defined in the class. We identify the attributes involved in implementing the action and check whether the attributes are defined in the class. An undefined attribute leads to attribute missing inconsistency. If an attribute is undefined, the vital information related to the attribute such as type, visibility etc. shall be missing and this leads to inconsistency.

B. Attribute Specification Inconsistency

An attribute is defined in terms of a set of properties such as name, type, visibility, id, class name, multiplicity, changeability, ordering and initial value. Attribute specification inconsistency occurs when there are incompletely specified attributes, i.e. the attribute is mentioned in the class, but the properties of the attribute are not completely specified. We have defined a consistency index called Attribute Specification Index, ASI to compute the consistency of the particles. ASI is defined as the ratio of the fitness value of the particle to the fitness value of the gbest particle.

$$ASI = f(Particle) / f(gbestParticle) \quad (3)$$

The class diagram is the only diagram that can be directly mapped to an object oriented programming language and hence incomplete specification of the attributes may lead to the development of erroneous software.

V. Consistency Checking of Attribute Definition using PSO

The principle of particle swarm optimization is applied to detect and fix inconsistencies in attribute definitions. The aim of our work is to optimize the fitness value of attributes to achieve consistency. ASI value is computed for each particle and it is also used as the stopping criteria of the PSO algorithm. Detection and correction of inconsistencies during the design stage of software development life cycle results in the generation of accurate code. Our method performs consistency check of the attribute definitions between class and activity diagrams by applying the principle of PSO and generates optimized diagrams by maximizing the fitness value of particles (attributes). Maximizing the fitness value improves the attribute specification.

A. Algorithm

The algorithm for implementing the PSO method for consistency checking of attribute definitions is outlined below [3].

Model the requirements using UML class diagram and activity diagrams.
  Parse the diagram and obtain diagram specification.
  Identify the attributes in the implementation of action in the nodes of the activity diagram.
  Check attribute missing inconsistency.
  Create new particles for each missing attribute.
  Initialize the search space with particles.
Initialize c1, c2, maximum number of iterations, maximum velocity, initial velocity, inertia weight.
Repeat
  Apply fitness function to each particle.
  Record pbest and gbest.
  Compute velocity of the particle

$$v[t+1] = w * v[t] + c1 * rand(0,1) * (pbest - present[t]) + c2 * rand(0,1) * (gbest - present[t])$$

  Update position

$$present[t+1] = present[t] + v[t+1]$$

  Compute ASI

$$ASI = f(Particle) / f(gbestParticle)$$

Until ASI=1 or maximum number of iterations is attained

The system modeled using class and activity diagrams are parsed to identify attributes and nodes in the diagram. Particles are created corresponding to each attribute defined in the class diagram. The nodes in the activity diagram and the attributes involved in the computation are identified. For each attribute identified in the activity diagram, we check for attribute missing inconsistency. If the attribute is missing, new particle with the attribute name is added to the search space. The attribute specification consistency of all the attributes is checked and ASI index is computed. The PSO algorithm is applied iteratively until the ASI value of all the particles are one or the maximum number of iterations is achieved.

The main components of PSO are a swarm of particles with each particle representing a candidate solution. Elements of the particle represent the parameters to be optimized [27]. The PSO algorithm initializes the search space with particles, evaluates the fitness of particles, computes velocity and updates position of the particles.

B. Particle Initialization

To apply the PSO algorithm, an initial population of particles called swarm is to be considered. The type of particles depends on the problem we are trying to optimize [5] [7]. To perform consistency checking of the attribute definitions, attributes specified in the class diagram and activity diagram are defined as particles. A particle is represented as a nine tuple consisting of id, name, type, visibility, class name, multiplicity, changeability, ordering and initial value. Name refers to the name of the attribute, type represents the data type, visibility refers to the access specifier, and class name represents the name of the class in which the attribute is defined. Attribute missing inconsistency check is performed during particle initialization to identify missing attributes. New particles with the name of the attribute are created for each missing attribute [3].

C. Fitness Function

The fitness value, computed by applying a fitness function to each particle, is a measure of consistency of the particle. A particle is defined using a set of parameters or properties. The fitness value is computed as a function of the parameters in the representation of a particle. Each parameter is assigned a weight according to its significance and the fitness function computes the weighted sum of the parameters. The goal of our algorithm is to maximize the fitness function. A high value of fitness function implies that the particle is consistent [3].

The fitness function is applied to all particles and the fitness value is computed in all iterations. For all particles, we compute ASI, the attribute specification index. ASI provides a measure of how consistent the attribute specification is. A specification index value less than one indicate inconsistency.

D. Position / Velocity Adjustment

If the ASI index of an attribute returns a value less than one, the particle is inconsistent. If the properties of the particles are not defined completely, the particle is inconsistent. The velocity and position of inconsistent particles are updated based on the values of present, pbest and gbest. Consistency is achieved by moving the parameters of the particles in the search space to their respective positions according to PSO equations (1) and (2). To apply the PSO algorithm for solving model consistency problem, we define velocity as the number of parameters to be added to a particle in each iteration and position as the total number of defined parameters of the particle. The process is repeated until an optimum solution that satisfies the consistency criteria is achieved or maximum number of iterations is reached.

E. Case Study

A project Library has been implemented to perform consistency check. The class diagram for the project Library is illustrated in figure 1. The inconsistency between the diagrams is demonstrated with help of a class diagram and an activity diagram. The project consists of a class diagram with six classes Book, Member, Librarian, Issue, Student and Faculty and an activity diagram for the method issueBook() in the class Librarian. Each class has its own attributes and methods.

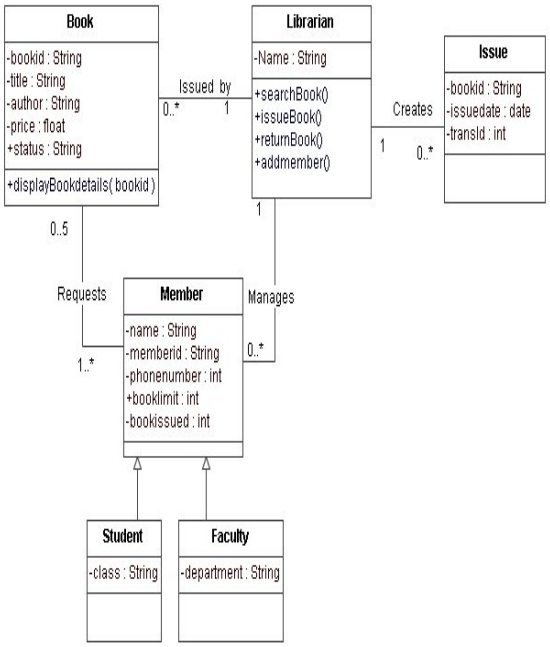

Figure 1. Class diagram Library

The attributes of the class Book are bookid, title, author, price and status. The data types of *bookid, title, author and status* is String and the data type of *price* is float. The visibility of the attribute *status* is public whereas the visibility of all other members is private. The class has one method *displayBookdetails()* with one argument *bookid*.

The class Librarian has only one attribute *name* of type String. The methods are *searchBook()* to search for a book, *issueBook()* to issue the book to a particular member, *returnBook()* to handle the books returned by the member and *addmember()* to add a new member to the library. The class Issue stores the issue details of books and has three attributes *bookid, issuedate* and *transid*. The data type of the attribute *bookid* is String, *transid* is integer and *issuedate* belongs to the data type date.

The class Member defines the attributes of members of the library. The attributes are *name* to store the name of the member, a unique *memberid* to store the id of the member, *phonenumber* to store phone number, *booklimit* to indicate the maximum number of books that can be issued to a member and *bookissued* to indicate the maximum number of books that has been issued to the member.

The classes Book and Librarian are connected by an association with name *issued by* and the association name between Librarian and Issue is *creates* since the issue process is created by the Librarian. The Librarian *manages* Member and a member can *request* for a Book. Members can be of two types: Faculty or Students. A generalization relationship exists between the two classes and the Member class.

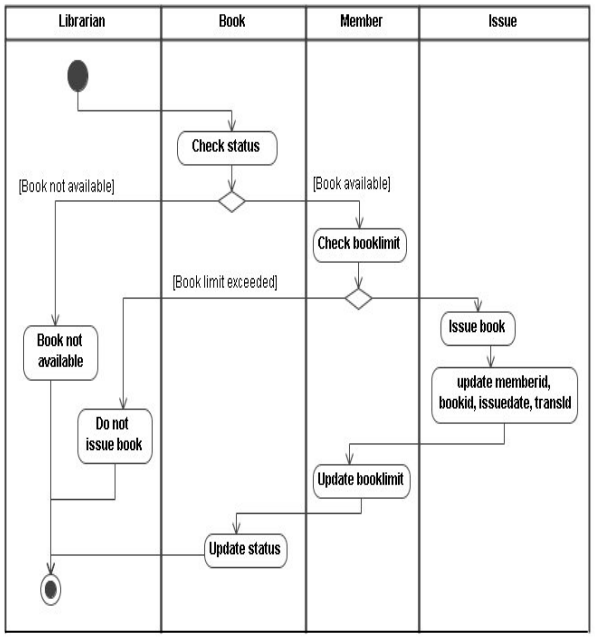

Figure 2. Activity diagram for issueBook()

The activity diagram for the method issueBook() method in the class Librarian is depicted in figure 2. The activity diagram provides a diagrammatic representation of the sequence of actions involved in the issuing a book to a member. The issueBook() method is initiated by the Librarian. The activity diagram checks the status of the book. Status is an attribute defined in the class Book. The attribute specifies whether the book is available or issued. If the book is not available, the issue process ends. If the book is available, the book limit of the member is checked in the class Member. Since the attribute is accessed by a nonmember function, the visibility of the attribute is specified as public. If the book

limit has been reached, the book is not issued. If not, the book is issued to the member and the transid, memberid, bookid and issuedate are updated in the Issue class. The bookissued attribute is updated in the Member class and the status of the book is updated in the class Book and the issueBook() method terminates.

The node *'update memberid, bookid, issuedate, transid'* in the class Issue refers to four attributes *memberid, bookid, transid and issuedate* whereas only three attributes *bookid, transid and issuedate* are defined in the class Issue. This leads to attribute missing inconsistency. Attribute missing inconsistency check detects the missing attribute and a new particle with name *memberid* is created in the class Issue. The algorithm performs attribute missing inconsistency check on all the nodes in the activity diagram. New attributes or particles are created for all missing attributes. During iteration of the PSO algorithm, ASI value of the *memberid* particle has a value less than one since the parameters are not defined. Attribute specification inconsistency check is performed on all particles in all iterations. The properties of the attributes are updated in all iterations depending on the velocity value, resulting in a change in the position. The PSO algorithm is iterated repetitively until all the particles are made consistent or the termination condition is reached.

VI. Conclusion

We have developed a method to improve the UML class attribute definitions by applying the principle of PSO. UML diagrams model the different aspects of the system and intra-model consistency should be achieved among the diagrams of the same model. In our method the static aspects are modeled using class diagrams and dynamic aspects using activity diagrams. We employ the particle swarm optimization algorithm to perform the consistency check on class and activity diagrams of the same model. The attributes are treated as particles and the particle is represented as a tuple consisting of different parameters. The fitness function analyzes the attribute definitions to compute the fitness value of the particle. The attribute specification index, ASI evaluates each particle with gbest particle. The ASI value is also used as the stopping criteria. The position and velocity of the inconsistent particles are updated. Object oriented programming languages are centered on the concept of classes and class diagrams can be mapped directly to an object oriented programming language and hence detection of attribute missing inconsistency or attribute specification inconsistency plays a significant role in the development of accurate software. Large complex systems require many diagrams to model the system and inconsistencies if undetected results in the development of software with errors. Correction of errors in the software may turn out to be costly and time consuming. Manual detection of inconsistencies is also time consuming and error prone. The advantages of our method are that it uses the simple concept of PSO to detect and fix inconsistencies in attribute definitions and the method can be applied in the design stage of software development thereby saving cost, time and effort and results in the production of more accurate code.

References

[1] Floreano. Dario, Claudio. Mattiussi. *Bio-inspired artificial intelligence: theories, methods, and technologies*, MIT press, 2008.

[2] Kennedy. J, Eberhart. R, " Particle Swarm Optimization". In *Proceedings of IEEE International Conference on Neural Networks. **IV***, pp. 1942–1948, 1995.

[3] George. Renu, Philip. Samuel. "Particle Swarm Optimization Method Based Consistency Checking in UML Class and Activity Diagrams", In *Innovations in Bio-Inspired Computing and Applications*, Springer International Publishing, pp. 117-127, 2016.

[4] Liu. Wen. Qian, Steve Easterbrook, John Mylopoulos. "Rule-based detection of inconsistency in uml models". In *Workshop on Consistency Problems in UML-Based Software Development*, Vol. 5, 2002.

[5] Particle Swarm Optimization, http://www.swarmintelligence.org

[6] Van. Der. Straeten. Ragnhild, Tom. Mens, Jocelyn. Simmonds, Viviane. Jonckers. "Using description logic to maintain consistency between UML models". In: *«UML» 2003-The Unified Modeling Language. Modeling Languages and Applications*, Springer Berlin Heidelberg, pp. 326-340, 2003.

[7] O'Keeffe. Mark, Mel. O. Cinnéide. "Towards automated design improvement through combinatorial optimization". In *Workshop on Directions in Software Engineering Environments*, Edinburgh, Scotland, UK. 2004.

[8] Blanc. Xavier, Isabelle. Mounier, Alix. Mougenot, Tom. Mens. "Detecting model inconsistency through operation-based model construction". In*: ACM/IEEE 30th International Conference on Software Engineering (ICSE'08)*, pp. 511-520. IEEE 2008.

[9] Dubauskaite. R, O. Vasilecas. "Method on Specifying Consistency Rules among Different Aspect Models, expressed in UML". *Elektronika ir Elektrotechnika*,19(3), pp.77-81, 2013.

[10] Egyed. Alexander. "Instant consistency checking for the UML". In *Proceedings of the 28th international conference on Software engineering*, pp. 381-390, ACM 2006.

[11] Van Der Straeten., Ragnhild., Jocelyn Simmonds., Tom Mens.: "Detecting Inconsistencies between UML Models Using Description Logic". *Description Logics*, 81, 2003.

[12] Saini. Dinesh Kumar, Yashvardhan. Sharma. "Soft computing particle swarm optimization based approach for class responsibility assignment problem". *Soft Computing*, 40(12), 2012.

[13] Ducatelle. Frederick, Gianni. A. Di. Caro, Luca. M. Gambardella. "Principles and applications of swarm intelligence for adaptive routing in telecommunications networks", *Swarm Intelligence*, 4(3), pp. 173-198, 2010.

[14] Shamshiri. Meysam, Chin. Kim. Gan, Yusoff. Mariana, Mohd. Ruddin, AbGhani. "Using Particle Swarm Optimization Algorithm in the Distribution System Planning", *Australian Journal of Basic and Applied Sciences* 7(3), pp. 85-92, 2013.

[15] C. Revathi, M. Mythily. "A Uml/Marte Detection of Starvation and Deadlocks at the Design Level in Concurrent System", *Int. J. Computer Technology & Applications*, 4(2), pp. 279-285, 2013.

[16] Nytun. Jan. Pettersen, Christian. S. Jensen. "Modeling and testing legacy data consistency requirements". In: *«UML» 2003-The Unified Modeling Language. Modeling Languages and Applications*, Springer Berlin, Heidelberg, pp. 341-355, 2003.

[17] Engels. Gregor, Jochem. M. Küster, Reiko. Heckel, Luuk. Groenewegen. "A methodology for specifying and analyzing consistency of object-oriented behavioral models". In *ACM*

SIGSOFT Software Engineering Notes, 26(5), pp. 186-195. ACM 2001.

[18] Kennedy. J, Kennedy. J. F, Eberhart. R. C, Shi. Y. *Swarm intelligence*, Morgan Kaufmann, 2001.

[19] Peram. Thanmaya, Kalyan. Veeramachaneni, Chilukuri. K. Mohan. "Fitness-distance-ratio based particle swarm optimization". In *Proceedings of the Swarm Intelligence Symposium (SIS'03)*, pp. 174-181, IEEE 2003

[20] Bai. Qinghai. "Analysis of particle swarm optimization algorithm", *Computer and information science*, 3(1), pp. 180, 2010.

[21] Hu. Xiaohui, Russell. C. Eberhart. "Adaptive particle swarm optimization: detection and response to dynamic systems". In *wcci*, pp. 1666-1670, IEEE 2002.

[22] De. Souza. Luciano. S, Pericles. B.C. de. Miranda, Ricardo. B. C. Prudencio, Flavia. De. A. Barros. "A multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort". In *23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 245-252. IEEE 2011.

[23] Huzar. Zbigniew, Ludwik. Kuzniarz, Gianna. Reggio, Jean. Louis. Sourrouille. "Consistency problems in UML-based software development". In *UML Modeling Languages and Applications*, Springer Berlin, Heidelberg, pp. 1-12, 2005.

[24] Bansal. J. C, Singh. P. K, Saraswat. M, Verma. A, Jadon. S. S, Abraham. A. "Inertia weight strategies in particle swarm optimization". In *Third World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pp. 633-640. IEEE 2011.

[25] Blondin. James. "Particle swarm optimization: A tutorial". http://cs.armstrong. edu/saad/csci8100/pso tutorial.pdf , 2009.

[26] Shi. Y, Eberhart. R. "A modified particle swarm optimizer". In *Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence*, pp. 69-73, IEEE 1998.

[27] "Particle Swarm Optimization", Computational Intelligence: Second Edition, http://ci.cs.up.ac.za/chapter16.pdf

[28] Rasch.Holger, Heike. Wehrheim. "Checking consistency in UML diagrams: Classes and state machines". In *Formal Methods for Open Object-Based Distributed Systems*, Springer Berlin Heidelberg, pp. 229-243. 2003.

[29] Chiorean. Dan, Mihai. Paşca, Adrian. Cârcu, Cristian. Botiza, Sorin. Moldovan. "Ensuring UML models consistency using the OCL Environment". *Electronic Notes in Theoretical Computer Science*, 102, pp. 99-110, 2004.

[30] Briand. Lionel. C, Yvan. Labiche, L. O. Sullivan. "Impact analysis and change management of UML models". In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pp. 256-265, IEEE 2003.

[31] Soeken. Mathias, Robert. Wille, Mirco. Kuhlmann, Martin. Gogolla, Rolf. Drechsler. "Verifying UML/OCL models using Boolean satisfiability". In *Proceedings of the Conference on Design, Automation and Test in Europe, European Design and Automation Association*, pp. 1341-1344, 2010.

[32] Szlenk. Marcin. "Formal Semantics and Reasoning about UML Class Diagram". In *International Conference on Dependability of Computer Systems (DepCos-RELCOMEX'06)*, pp. 51-59, IEEE 2006.

[33] Lucas. Francisco. J, Fernando. Molina, Ambrosio. Toval. "A systematic review of UML model consistency management", *Information and Software Technology*, 51(12), pp. 1631-1645, 2009.

[34] Mens. Tom, Ragnhild. Van. Der. Straeten, Jocelyn. Simmonds. "*A framework for managing consistency of evolving UML models*", Software Evolution with UML and XML, pp.1-31, 2005.

[35] Engels. Gregor, Reiko. Hecke.l, Jochen. Malte Küster. "Rule-based specification of behavioral consistency based on the UML meta-model". In *≪ UML≫ 2001-The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, Springer Berlin Heidelberg, pp. 272-286, 2001.

[36] Egyed. Alexander, Emmanuel. Letier, Anthony. Finkelstein. "Generating and evaluating choices for fixing inconsistencies in UML design models". In *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008)*, pp. 99-108, IEEE 2008.

[37] Evans. Andy. S. "Reasoning with UML class diagrams". In *Proceedings of the 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques*, pp. 102-113, IEEE 1998.

[38] Genero. M, Piattini. M, Calero. C. "A survey of metrics for UML class diagrams", *Journal of object technology*, 4(9), pp. 59-92, 2005.

[39] Perdita. Stevens., Rob Pooley. *Using UML software Engineering with Objects and Components*, Pearson Education, 2003.

[40] Blanc. Xavier, Alix. Mougenot, Isabelle. Mounier, Tom. Mens. "Incremental Detection of Model Inconsistencies Based on Model Operations". In *CAiSE*, (9), pp. 32-46, 2009.

[41] Kaneiwa. Ken, Ken. Satoh. "On the complexities of consistency checking for restricted UML class diagrams", *Theoretical Computer Science*, 411(2), pp. 301-323, 2010.

[42] Kiranyaz. Serkan, Jenni .Pulkkinen, Moncef. Gabbouj. "Multi-dimensional particle swarm optimization in dynamic environments", *Expert Systems with Applications* 38(3), pp. 2212-2223, 2011.

[43] Diskin. Zinovy, Yingfei. Xiong, Krzysztof. Czarnecki. "Specifying overlaps of heterogeneous models for global consistency checking". In *Proceedings of the First International Workshop on Model-Driven Interoperability*, pp. 42-51, ACM 2010.

[44] Zhan. Shaobin, Hongying. Huo. "Improved PSO-based task scheduling algorithm in cloud computing", *Journal of Information & Computational Science*, 9(13), pp. 3821-3829, 2012.

Author Biographies

Renu George acquired her M.Tech degree in Computer and Information Science from Cochin University of Science and Technology and B.Tech degree in Computer Science and Engineering from Kerala University. She is an Assistant Professor in College of Engineering, Chengannur, Kerala, India and Ph.D research scholar in Cochin University of Science and Technology. Her areas of interest include compiler design, automata theory, UML modeling and design, software engineering and artificial intelligence.

Dr. Philip Samuel is an Associate Professor in Information Technology at Cochin University of Science and Technology, Kochi. He holds a Ph.D degree in Computer Science & Engineering from Indian Institute of Technology, Kharagur and an M.Tech degree in Computer and Information Science from Cochin University of Science and Technology. He has several research publications in international conferences and journals. His research interests include Artificial Intelligence, Distributed Computing, UML Modelling and Design and Software Engineering.