Volume 4, Issue 4, April 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

MARTE Genetic Algorithm for Uncovering Scenarios Leading to Static and Dynamic Data Races in Concurrent Systems

K. Venkata Siva Rami Reddy[1]
Department of Computer Science and Engineering
Gitanjali Institute of Technology
Proddatur, Andhra Pradesh, India.

Dr. Karanam Madhavi[2]
Department of Computer Science and Engineer
Nalgonda Institute of Technology and Science
Nalgonda, Andhra Pradesh, India.

G. Ramesh[3]
Department of Computer Science and Engineering
JNTU College of Engineering Anantapur
Anantapur, Andhra Pradesh,India .

Abstract: Identifying the concurrency problems early in the design process is most important. Because, these problems (Deadlock, Starvation and Data Race) caused to disturbance or damage when building larger and more complex systems. The recent trend is Model Driven Development (MDD), so there is a method used for the detecting concurrency problems, which is based on design models articulated in Unified Modelling Language (UML). The UML notation is not enough to model a system for s given purpose, so the notation is extended via profiles. The system aim is to develop a scalable integrated method to tailor all (static and dynamic) concurrency problems. This is achieved through three steps. First one is extracting all relevant concurrency information using UML/Modelling and Analysis of Real-Time and Embedded Systems (MARTE) model diagrams. Second one is detecting all concurrency faults by using the search-based technique. Third one is demonstrating scalability in terms of fault detection. Apart from Dead lock and Starvation, also detecting one more concurrency problem such as Data Races, in Data Races find the static and dynamic dataraces.

Keywords: Deadlock, Starvation, Data Races, MDD, UML, MARTE, GENETIC ALGORITHM.

1. INTRODUCTION

Concurrency problems should be recognized before in the design procedure. If not, this is made increasingly hard as larger and more complex in concurrent systems. The recent fashion in the direction of Model Driven Development (MDD) [4] [1][10][11], the alternative of using UML models and their extensions as a foundation of concurrency data at the design level is regular and realistic. These concurrency properties should not require additional modelling or a high learning curve on the element of the designers, or should at least diminish it. When the UML [5] notation is not sufficient to absolutely model a system for a given rationale, the UML notation is extended via profiles. Therefore specific significance is the standardization of the MARTE [1] profile that deals with domain specific characteristic of real-time, concurrent system modelling. Aim is to develop a general, automated approach that can be easily tailored to several types of concurrency faults (such as deadlocks, starvation, data races), and the system can be straightforwardly incorporated into a Model Driven Architecture (MDA) approach [4], [5], [1]. This approach relies on a genetic algorithm (GA) [2] that is tailored to different types of concurrency errors. The UML is a designing language, which is used for visualizing system, specifying and constructing the artifacts of software intensive systems [3], [5]. Nowadays, it is regard as the standard for object-oriented modelling. UML [3] allows modelling various aspects of complex systems. The models those are designed by UML present some vagueness's and contradictions as mentioned in several papers. This limitation can lead to contradiction within the developed models. The usage proper techniques, particularly in the case of complex systems, it gives notable advantages, like a simpler design without ambiguities. Simultaneous programming is an authoritative hypothesis where actions can be executed alongside [2]. However, this method of programming has exact troubles. The simultaneous threads performing on the similar resources can guide to not needed and unpredicted conditions. For example, deadlocks, live locks or data inconsistencies may occur. Wegner explains the universal idea of lively objects. These are activated when receiving a message. These objects may be previously executing when receiving a message. When discussing about the external concurrency, it mean that when two active objects implementing on the same resource with a single thread, those will be in sequential. Also, Building models of real-time application requires a modelling language that enables the description of the specific features inherent to real-time domain [8]. Actually real-time applications have some qualitative features such as deadline and period, as well as quantitative features related to communication, concurrency and behaviour aspects. The modelling language must at least provide the syntax (modelling concepts) to represent such features [3]. However, semantics associated to those concepts has to be precisely defined and unambiguous in order to ensure the excitability of real-time applications models [8]. Actually, in order to reduce the impact of changes on the system, the system should be able to validate at early stages of the development process, functional and extra functional properties of the application. A way to achieve this goal is to make application models executable [3].

A model is considered executable when it can be functioning on a hardware execution unit. A model can be operated if its behavior is clearly and completely specified. Hence, in order to enable a real-time application model excitability, that model has to be specified in a language that provides on the one hand all the concepts needed with a precise semantics and on the other hand, the concepts to ensure completeness of behavioral models.

2. RELATED WORK

The system move towards covers quite a few fields of concurrent information, such as deadlock, starvation and shared resources.

2.1 Starvation Detection

Starvation condition is the state where a procedure carries on being rejected a resource, even though the resource is being furnished to some other process to its need. Starvation state arise and treating them by stopping new processes from obtaining resources.

2.2 Genetic Algorithm

A Genetic Algorithm may be decayed into the following steps:

- ✓ Starting population is to be created. Frequently a set of unsystematic chromosomes are created [2].
- ✓ Do again the below steps awaiting some extinction measure is met:
- ✓ Estimate each chromosome using a fitness function.
- ✓ Choose the pairs of chromosomes using some strategy such as random selection or fitness biased techniques [6], [7].
- ✓ Apply crossover on the couples of chromosomes selected and mutation on personages [1], [2], [6].
- ✓ Create a new population by changing a part of the novel population with the chromosomes 'produced' in the earlier step [1], [6].

2.3 Chromosome Representation

Genetic Algorithm starts with foremost population whose elements are called as chromosomes. The chromosome consists of a precise number of variables which are called genes. In order to estimate and grade chromosomes in a population, a fitness function stand on the target function should be defined. A Chromosome is a representation of an individual solution for a specific problem [6]. You will have to redefine the Chromosome representation for each peculiar problem, along with its fitness, mutate, reproduce, and seed methods. Chromosomes are the central objects in a genetic algorithm. Chromosomes are defined by the GA Chromosome class in this library.

2.4 Crossover Operator

Crossover picked genes from parent chromosomes and generates a fresh offspring. The easiest way how to do this is, to select randomly some crossover point and anything earlier than this point duplicate from an initial parent and then anything after a crossover point replica from the second parent. Crossover operator targets to exchange the data and genes between chromosomes [2]. Therefore, crossover operator integrate two or more parents to regenerate fresh children, then, one of these children may gather all good characteristics that survive in his parents. Crossover operator is not peculiarly useful for all parents but it is useful with probability pc which is generally fixed equal to 0.6 [6], [7], [1]. Crossover operator plays a leading role in GA, so defining a particular crossover operator is highly required to get a best performance of GA. A crossover operator is used to re arrange two strings to get a best string. In crossover manipulation, recombination process generates dissimilar individuals in the consecutive generations by adding material from two individuals of the earlier generation. In reproduction, adequate strings in a population are probabilistically assigned an overweight number of duplicates and a mating pool is made [7]. It is to be noted that during the propagation phase no new strings are formed. In the crossover operator, fresh strings are generated by interchanging information among strings of the pairing pool.

2.5 Mutation Operator

Mutation is a genetic operator, which involves the process of changing one or multiple gene values in a chromosome from its early state. This process can result in completely new gene values being combined to the gene pool. By those new gene values, the genetic algorithm is capable to attain better solution than in the past possible [6]. Mutation is the considerable part of the genetic search as helps to prevent the population at any local optima. Mutation probability says how frequently will be parts of chromosome mutated. If mutation is not existing, descendent is taken after crossover without any alters. If mutation probability is 100%, whole chromosome is altered [1], if it is 0%, not anything is modified. Mutation is made to stop falling GA into local tremendous [7], but it must not happen very often, because then GA will practically change to random search.

2.6 Objective Function

The Objective Function is used to find the how much time the thread can be waited or starved to get the regular access to gain the resource. This function is used in the starvation detection phase to know the time to get the resource by the thread.

2.6.1. Starvation Detection

Starvation occurs when a scheduler process rejects to provide a specific thread any quantity of a specific resource. If there are numerous high-priority threads, a weaken priority thread may be starved. This can have unenthusiastic effect, though peculiarly when the lesser main concern thread has a lock on some resource.

Starvation is a situation, which occurs owing to deadlock occurs. In this one of the action is rolled back to come out from deadlock situation, so that all other processes can continue advance for their completion. If an identical process is choose as dupe for rollback repeatedly then this is known as starvation. The system uses this premise to develop an appropriate fitness function. Because, threads wait for driveway to a resource in a wait queue, we also need to analyse the wait queue of the target lock across the time interval. If the wait queue of the point lock be appropriate empty, then the target thread passage the lock and there is no starvation. The fitness function is weighted such that the longer the target thread spends waiting on the target lock, the superior its fitness.

$$f(c)= \sum_{i=\text{startTime}}^{\text{endTime}} \begin{cases} 1, \text{ if } i=0 \text{ and } A \in \text{execThreads}_i \text{ and } A \in \text{waiting Threds;} \\ i, \text{ if } A \in \text{exec Threads}_i \text{ and } A \in \text{waiting Threads}_i; \\ 0, \text{ otherwise;} \end{cases}$$

The variable A represents the target thread. Variables start Time and end Time denote the time interval start and end times, respectively. The group of threads executing within the target lock at time unit i is denoted exec Threads and waiting Threads is the set of threads waiting for access to the target lock at time i. The sets exec Threads and waiting Threads are obtained after scheduling threads and are calculated for every time unit of the time interval. This aims that before a fitness value is connected with a chromosome, the execution of threads, as named by their access times to locks in the chromosome must be scheduled by a scheduler. According to the defined fitness function, the only conditions that would result in starvation situations are ones where the target thread is waiting on the target lock at the end of the time interval (i.e., A waiting threads end Time). This is the termination criterion used to determine the presence of starvation.

2.7 Deadlock Detection

The deadlock problem has proved a popular area of research for many years. The earlier results are largely informal, and endure from the lack of a adequate underlying mathematical model. There are many examples exist to specify the deadlock situation, which may be defined as the stable blocking of a group of processes that either struggle for system resources or converse with everyone. Obtain two processes, P and Q, as an example shown in Fig 1. Both processes are compel resources, A and B. They execute correspondingly in the subsequent steps:

Clearly, if P gains A at the same time Q gains B, a deadlock occurs since neither of both can proceed to gains the other resource they need. If we observe this example and many others, we may discover the subsequent constraints that must be current for a deadlock to be probable:

The main objective is to illuminate that we use the same kinds of inputs and that the main difference with starvation detection is the fitness function.

The fitness function in possesses a number of characteristics:

1. Deadlocks engage minimum of two waiting threads. The fitness of plots where minimum two threads are waiting on locks is always greater than the fitness of situation, where no thread or atleast one thread is waiting.
2. The fitness function is determined by the total number of locks enabled, i.e., an additional thread implemented in the lock should improve the fitness value.
3. The fitness function is determined by the total number of threads remain in lock state which mean an additional thread waiting to acquire a lock should increase the fitness.

Process P:	Process Q:
……	……..
Get A	Get B
Get B	Get A
……	……..
do_something()	do_something_else()
…..	…….
Release A	Release B
Release B	Release A
…..	……

Fig 1: Deadlock occurring process

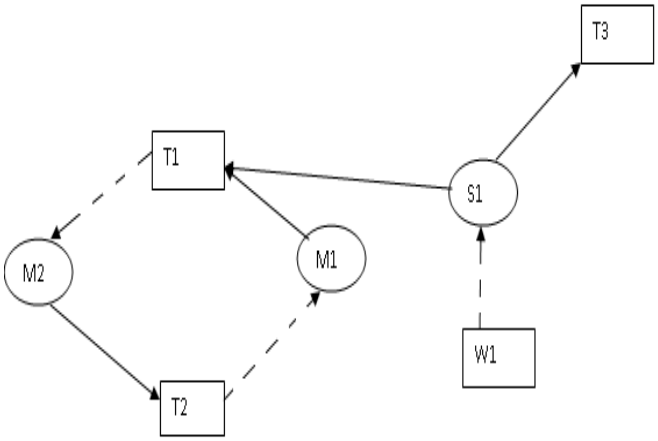

Fig 2: Deadlock and starvation depicted in an RAG

2.8 MARTE Profile

This section describes how can fulfill the first of three steps in achieving the system goal. It describes the feasibility of extracting all relevant concurrency information from UML/MARTE design diagrams. First describing the present, in UML. Then we show how the missing concurrency aspects can then be extracted from MARTE models [1]. In UML, the active objects have their own thread of lead, and can be concerned with as simultaneous threads. The UML standard extensions, such as the MARTE profile, afford method to model detailed information concerning to concurrency. The MARTE profile is replacement to the Schedulability, Performance and Time (SPT) profile [1]. MARTE is geared toward both the real-time and embedded system domains. The profile is approximately divided into three subdivisions: MARTE Foundation Model, MARTE design model and MARTE analysis model. The anterior models various features of real-time and embedded systems, while posterior allows the annotation of models for system analysis purposes. Both compartments are based on a common foundation, which is MARTE foundation, which describes the time concepts and use of concurrent resources. Much like its SPT antecedent, the MARTE profile is modular in structure, allowing users to select the exact subsets needed for their applications.

3. PROPOSED WORK

The proposed system tailored other or one more concurrency problem such as Data races (shared resources). So, for finding the data race problems, gathering the all the relevant concurrency information using UML/MARTE profiles and tailor the generic algorithm to the data races.

3.1. Data Races

Data races, a specific type of race conditions, are quite common in concurrent systems. An anomalous behavior due to unexpected critical dependence on the relative timing of events. These types of faults are due to unsynchronized access to a same memory location.

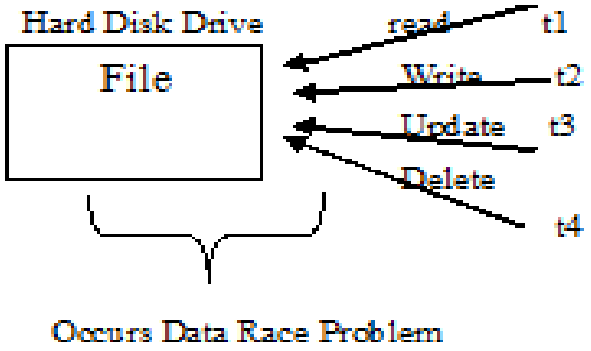

Fig 3: Representation of Data Race Problem

The system also assesses the performance of our approach on systems with varying structures to establish the particular characteristics of the system. Results of the detection rate of data races are presented in Table 1. The proposed technique is capable of detecting data races in Dining Philosophers, but with very different probabilities. We observe that GA does better: 34% detection rate. This confirms that where the search space is large and complex, GAs is known to yield much better results. Here, static data races considered based on only timing, but the dynamic data races were considered timing along with the time type of the thread that we apply on the file which is represent in fig 3.

Table 1: MARTE mapping Tags

Concept	MARTE Stereotype/Tag	MARTE sub-profile
Thread	<<SwConcurrentResource>>	SRM::SW_Concurrency
Lock	<<SwMutualExclusionResource>>	SRM::SW_Interaction
Lock acquire	<Acquire>>	GRM::ResourceTypes
Lock release	<<Release>>	GRM::ResourceTypes
Lock capacity	<<SwMutualExclusionResource>>	SRM::SW_Interaction
Lock accessrange	<<gaStep>>/interOccTime|	GQAM::GQAM_Workload

4. RESULTS

In large, complex search spaces, where few concatenations afford data races, the GA yields significantly higher detection probabilities than other techniques. Because, these probabilities for every single run can still remain low, the GA must be run as many times as possible, given time constraints, to obtain the highest possible overall probability of detecting data races. The following table describes the result comparison between previous performance and current performance of GA on Dining Philosophers problem.

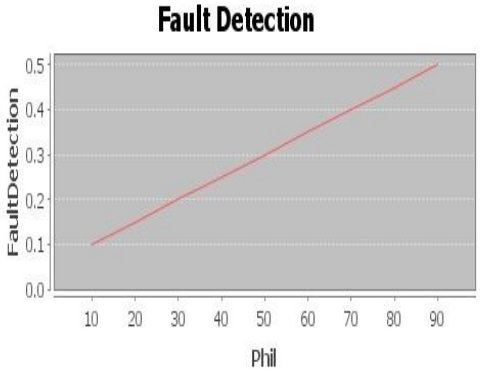

Fig 4: Scalability in terms of Fault Detection

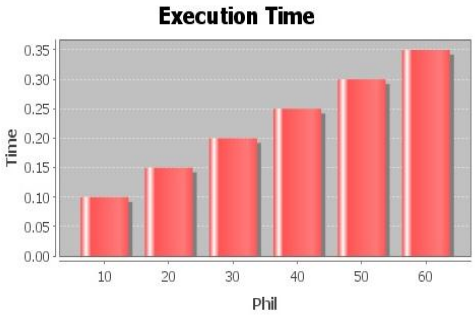

Fig 5: Scalability in terms of Execution time in Seconds

The Fig 4. shows the detection of faults while performing GA on dining philosopher's problem. These yield the better detection rate than the earlier result. In the same way the Fig 5 represents the execution time of GA on Dining philosopher's problem and it shows the improvement in speed of execution.

Table 2 Performance of GA on dining philosophers

DINING PHILOSOPHERS PROBLEM				
Fault Type	Dynamic Data Races	Static Data Races	Deadlock	Starvation
Search Space	$1.9*10^{119}$	$1.9*10^{119}$	$1.9*10^{119}$	$3.9*10^{232}$
Detection Runs	49/50	48/50	22/25	100/100
Total Runtime	01:02:02:255	01:02:02:255	2:34:34:482	0:00:02:558
Min. Runtime	0:00:08:020	0:00:08:020	0:00:20:238	0:00:00:029
Max. Runtime	0:35:10:546	0:35:10:546	0:43:11:998	0:00:01:010
Detection rate	96%	93%	90%	92%

5. CONCLUSIONS

Concurrency abounds in many software systems, where threads typically access many shared resources. If not addressed properly, such accesses can lead to concurrency errors, which may lead to appreciable system failure. The proposed system describes an approach, based on a tailored genetic algorithm (GA) search, for detecting one type of concurrency error: such as data races. The advances are based on the analysis of design delegations in UML completed with the MARTE profile. Since, the goal is to present an automated approach that can be operational in the perspective of model-driven and UML-based development.

REFERENCES

1. Marwa Shousha, Lionel C. Briand, Fellow, IEEE, and Yvan Labiche, Member, IEEE " UML/MARTE Model Analysis Method for Uncovering Scenarios Leading to Starvation and Deadlocks in Concurrent Systems" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 38, NO. 2, MARCH/APRIL 2012.
2. T. Back, "Self-Adaptation in Genetic Algorithms," Proc. European Conf. Artificial Life, pp. 263-271, 1992.
3. F. Schneider, "UML and Model Checking," Proc. Fifth Langley Formal Methods Workshop, 1999.
4. Stephen W. Liddle, " Model-Driven Software Development".
5. James Rumbaugh, Ivar Jacobson, Grady Booch, " The unified modelling language reference manual".
6. M. Dorigo and V. Maniezzo, "Parallel Genetic Algorithms: Introduction and Overview of Current Research," Parallel Genetic Algorithms: Theory and Applications, J. Stender, ed., pp. 5-43, IOSPress, 1993.
7. R.L. Haupt and S.E. Haupt, Practical Genetic A lgorithms. Wiley Interscience, 1998.
8. F. Schneider, "UML and Model Checking," Proc. Fifth Langley Formal Methods Workshop, 1999.
9. H. Gomaa, Designing Concurrent, Distributed, and Real-Time Applications with UML. Addison-Wesley, 2000.
10. D.C. Petriu, "Performance Analysis with the SPT Profile," Model-Driven Eng. for Distributed and Embedded Systems, pp. 205-224,Hermes Science Publishing Ltd., 2005.
11. A. Kleppe, J. Warmer, and W. Bast, MDA Explained—The Model Driven Architecture: Practice and Promise, Addison-Wesley, 2003.
12. OMG, UML Profile for Schedulability, Performance and Time Specification, http://www.omg.org/docs/formal/05-01-02.pdf.2005.

