A FRAMEWORK FOR SAFE COMPOSABLE TESTING MODEL FOR MULTIPLE APPLICATIONS TESTING ENVIRONMENT

[1]Ms.SMITHA.P.S, [2]Dr.N.SANKARRAM
[1]Anna University, Department of Computer Science and Engineering,VEC,Chennai,Tamil Nadu,India
[2]Anna University,Department of Computer Science and Engineering,RMKCET,Chennai,Tamil Nadu,India

E-mail: [1] smithaps.ap@gmail.com , [2]n_sankarram@yahoo.com

ABSTRACT

The objective of the work is to propose a safe composable and Stable Testing Model when multiple software components are to be integrated .The functionalities, interface, platforms are heterogeneous and hence the composition may give a correct outcome but not safe. In case of safety critical application and dynamic business transaction framework, a safe composable and stable execution of components is needed. In order to ensure Software Safety, A model is proposed where verification assist validation in ensuring complete Safeness across multiple product lines .This paper proposes a safe composable Testing Model which ensures the safe composability of components by constructing a Finite State Machine and thereby checking for safety by computational tree logic which assist in interface testing and generation of test cases for ensuring the complete coverage of test cases.

Keywords : *Safe , Composable , Finite State Machine, Coverage, Safety*

1. INTRODUCTION

The software safety becomes a challenge not only in the case of designing critical applications like Signaling Systems but also in the verification and validation of complex scientific systems. Software Safety should be compositional, compatible and should be able to model both hardware and software behaviour. The safety feature of the application software can be enhanced through a detailed safety requirement analysis that should specify the safety constraints whether they are implicitly time sensitive and or process sensitive. One of the essential criteria in any application is checking for Safety consideration. Safety requirements cover not only human safety, but also equipment and data safety. Human safety considerations include protecting the operator from moving parts, electrical circuitry and other physical dangers. There may be special operating procedures, which if ignored my lead to a hazardous or dangerous condition occurring. Equipment safety includes safeguarding the software system from unauthorised access either electronically or physically. An example of a safety requirement may be that a monitor used in the system will conform to certain screen emission standards. Safety Requirement are non functional requirement where faults are identified based on response time. For the system without a safety strategy implemented, the potential risk is high, because an unintended event may occur. The starting point for the development of any program is the expression of requirements and (or) specification of requirements placed by the customer or potential user upon the program to be created. The expression of requirements and (or) requirements specification must include the composition, content, and values of the results that are expected by the user, object, or system under certain conditions and initial data. Any deviation of these results from the requirements and reference values should be classified as an error in the program[6]. Seamless composability implies that a composition will have the desired beneficial properties, with no uncontrollable or unpredictable side effects. That is, the composed system will do exactly what it is expected to do —no more and no less. Composable system are more trustworthy than non-composable System[1]. Composability is the capability to select and assemble simulation components in various combinations into simulation systems to satisfy specific user requirements[2]. The defining characteristic of composability is the ability to combine and recombine components into different simulation systems for different purposes. The software components are self-checked based on the criticality of the information and data using temporal and computation tree logic[3][4]. The safety features may be different for different operational environment of a control application and so it is the limitation of the work to arrive a generic safety Testing model. It is impossible to create a complete model for testing many of the complex system, instead many of the methods which interact with each other can be considered. Model checking is a technique for automatically verifying correctness properties of finite state system. Model checking is done for checking logical properties of component coordination including deadlock, safety and liveness. An example of model checking is the formal specification of the system in propositional calculus and verifying it with the structure of the system. Temporal logic in model checking has the ability of reasoning with time constraints. Nowadays, testing is by far the most used technique for software verification in industry: it is easy to use and even when no error is found , it can release a set of tests certifying the (partial) correctness of the compiled system. In the case of safety critical software, in order to increase the confidence of the correctness of the compiled system, it is often required that the provided set of tests covers 100% of the code[3].The rest of the paper is organised as follow apart from Section 1 as Introduction, Section 2 specifies Safe Composable Testing Model(SCTM),Section 3 specifies Safe composition, Section 4 specifies Scenario for SCTM ,Section 5 specifies Model Checking, Section 6 specifies Incremental Integration Testing with FSM, Section 7 specifies Conclusion and Section 8 specifies Future Work.

2. SAFE COMPOSABLE TESTING MODEL(SCTM)

Fig1 illustrates the sequence of steps in framing a SCTM(Safe Composable Testing Model).If faults has to be avoided especially in ensuring Safety, measures have to be taken starting from requirement specification. Requirement in non-functional terms includes Composition and Compatibility. Verifying the requirements minimizes faults as well as cost, if proper checking is done at the initial phase. Nearly 60% of the faults emerges from the process of gathering and documenting requirement specification. There is a need especially in ensuring safety that there should be zero tolerance in propogation of any faults to later stages.

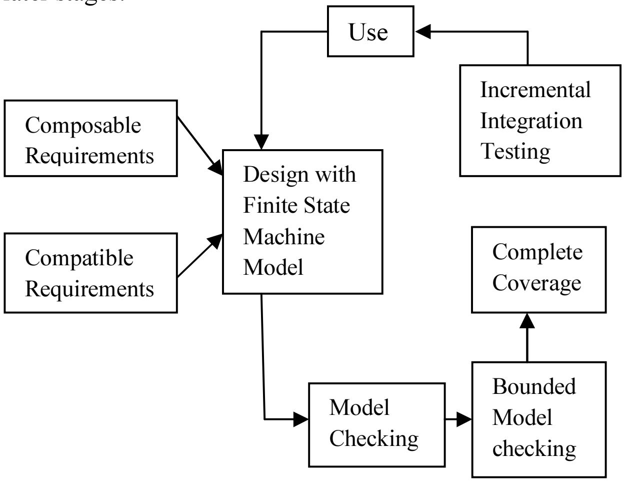


Fig 1: Safe Composable Testing Model

Hence, Verification ensures the Requirements have to be Correct, Complete, Consistent and Unambiguous. Composition of components and their interoperability depend on the correct specification of requirement. The composition of requirements can be expressed as FSM (Finite State Machine) Model. The finite State Machine can be expressed as nodes representing components and the edge representing the transition of the nodes to the next state based on the functional parameter. Model Checking is done to verify the safety property with FSM by mathematical representation of the temporal logics and computational tree logic of the model. Once the above steps are done, the requirements gathered are verified for safety property. Integration testing in the above model ensures the checking of interfaces between components in an timed distributed system. The interface between components is specified using FSM. Model Checker for the component interaction ensures the complete coverage of their interaction. Testing and Bounded model checking are two different methods used to verify the software or hardware system for finding faults. Testing is the process of identifying bugs in common behaviour of the system, whereas Bounded Model Checking(BMC) is the process of identifying bugs in uncommon behavior of the system. BMC is used here along with testing for the fact that if safety has to be ensured in the system then all of the faults related to it should be identified and this definitely ensures complete coverage of code.BMC Technique is used to achieve 100% Code Coverage.

3. SAFE COMPOSITION

The initial phase in the development of any Software Project activity is the establishment of requirements which is to be correct, complete and consistent. The requirements should include composition with values and results for the system or object under consideration.

Component C1 can be considered as a triple of C1 = (F, I, P, Ta) and it can be represented diagrammatically as in Fig 2.

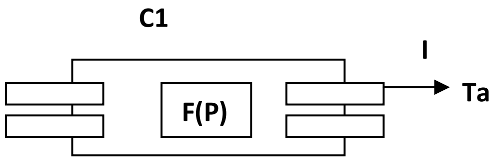


Fig 2: A Component

Where C1 is the component

F is the set of function in the component

P is the set of parameter passed to the component

T_a is the activation time.

$C = < (f_1,f_2,....f_i), I/O, (p_1,p_2...p_j), (Ta) >$

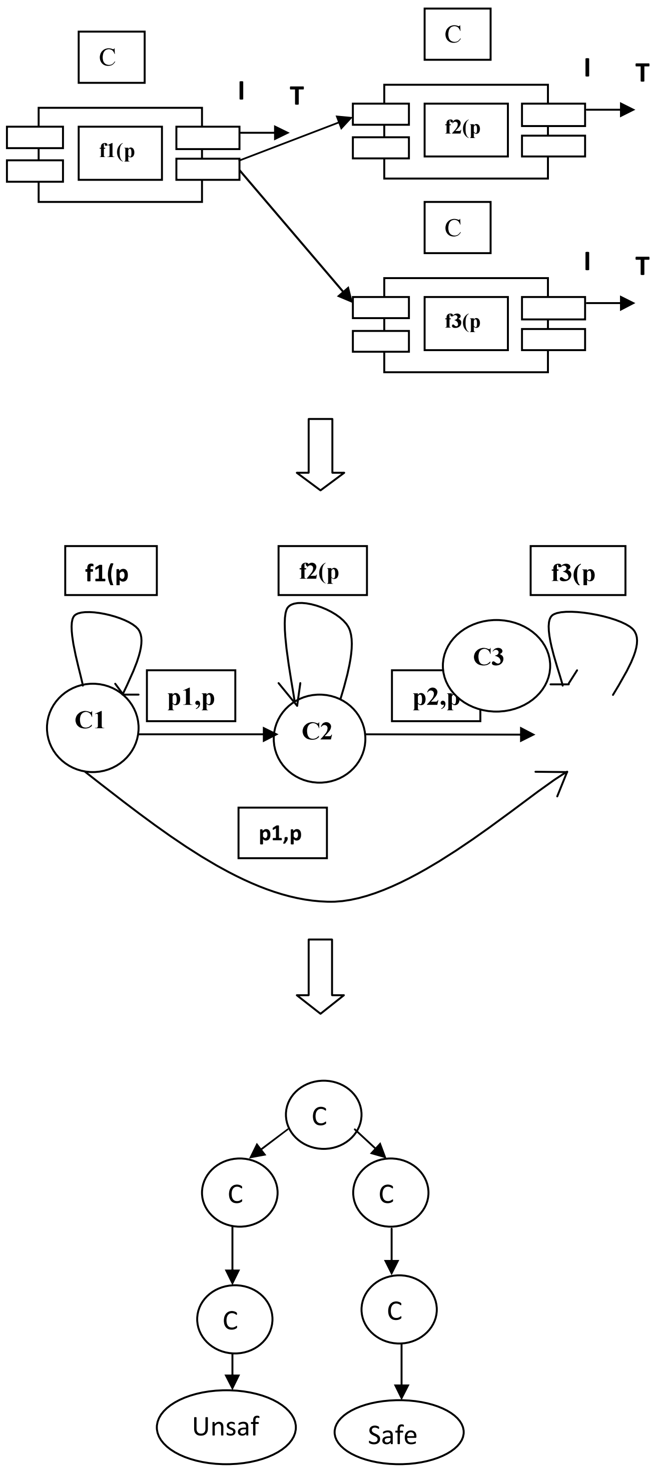
A component is considered as safe if it is composed towards expected functionality with the other needed components for the task submitted at that time. A component is considered unsafe, if it were composed with components of unexpected functionality. The composition activity is based on many read and selection techniques across the available interfaces during the point of time. It can also be seen that the untimely activation of the necessary component via its interfaces and idle time when a critical activity is being performed lead to unsafe not only for the software but also to the system where it is being activated. Composability can be viewed as the adherence or acceptance of the software module with other modules in or other platforms for effective parameter passing. Composability is dealt with both synchronous(Synchronous components is one in which all components in the system change their state variables simultaneously) and asynchronous components(In Asynchronous one component changes its state at each time point).Composition of its components can be expressed as timed automata which is used in real time application with timing requirements. Composability is represented as discrete timed automata (DTA) in which clocks take integer values. Formally, A clock constraint is a Boolean combination of atomic clock constraints in the following form : $x\sim c, x-y\sim c$, where ~ denotes $\leq, \geq, <, >$ or $=$, c is an integer and x,y are integer valued clocks. Let N be the set of integers with N+ for non negative integers. Let Lx be the set of all clock constraints on clock X.A discrete timed automata A is a tuple (Q, ∑, X, T), where Q is a finite set of states, ∑ is the input alphabet , X={x1,x2…..xn} is a finite set of integer valued clocks[2].

3.1 Composability And Compatibility of Requirements

Components are composed and if the interface of the components cannot be properly established then it adequately specifies the flaws in the requirement specification leading to 1)inadequate requirements 2) identification of inappropriate requirements and non-available requirements 3)Missing requirement. Compatibility with the composed components exposes the problem of 1)Multivendor Incompatability 2)Security Issues.

4. A SCENARIO FOR SCTM

Component which are interactable are represented as a graph and in the figure C1 component interactable with C2 and C3 with its functional elements. The component interaction is represented as a Finite Automata Model with its functional parameters deciding on the transition to the next state. With the Transition Model and Temporal Logic, a computational tree is drawn based on the assessment of safeness of Components as illustrated in Fig 3. If a function element f1 of C1 and function element f3 of C2 is compatible then the order of Composability can be represented as the Combinatorial equation as, *f1.O.C1 && f3.I.C2* where O is the output interface and I is the input interface are considered to be safe, otherwise it is unsafe. The graph representation of Components can be used for incremental interface testing which promote reusability and hence saving time and resources. Bounded Model checking is done with branch testing(BT) and statement testing(ST) and their coverage metrics(BC,SC) gives an indication of the Coverage of Test.



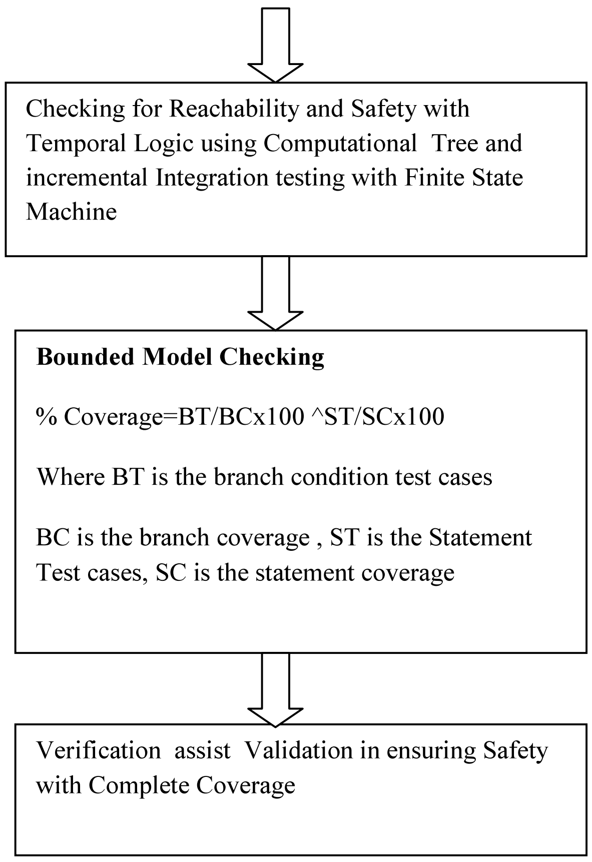

Checking for Reachability and Safety with Temporal Logic using Computational Tree and incremental Integration testing with Finite State Machine
Bounded Model Checking
% Coverage=BT/BCx100 ^ST/SCx100
Where BT is the branch condition test cases
BC is the branch coverage , ST is the Statement Test cases, SC is the statement coverage
Verification assist Validation in ensuring Safety with Complete Coverage

Fig 3: Scenario for SCTM

5. MODEL CHECKING

For verification of safety properties, a model checker is used, and properties to be checked are usually expressed in computation tree logic (CTL) [10,11]. CTL is a branching time temporal logic, extending propositional logic with temporal operators that express how propositions change their truth values over time. Model checking is done for the verification of properties in the System. The property which is to be checked in the system is Safety property. The safety critical product should satisfy its safety properties in all allowable configurations. The employed model checking approach can also be used to generate interesting scenarios for more detailed inspection with traditional testing and simulation[13].

6. INCREMENTAL INTEGRATION TESTING WITH FSM

Software for testing consist of several black-box components and which can be represented as state-based models of each component. These models are in the form of timed interface automata [7]. In general, an automatic test generation technique works by pursuing a set of test objectives identified after some (machine-readable) description of the software under test. Model-based test generators refer to models formalized out of the software specification,and devise test cases to exercise the behaviors represented in the models. For example, for a software system specified as a state machine, a test generator may attempt to generate test cases that execute all the state transitions of the state-machine model[12].Testing from timed interface automata identifies the communication between components. A set of test sequences is generated which checks communication between components by a predefined test criteria and Model checker. The number of states in the system grows as the number of components keep increasing. A partial model is generated which can be integrated with other components to form a complete model which emphasizes Incremental Integration Testing. The use of a model checker resolves timing and feasibility problems both on the component and on the inter-component level. A model checker has been used for both coverage analysis [8] and test sequence generation and use a subset of the Computational Tree Logic (CTL) [9].

7. CONCLUSION

A Stable Safety Testing Model has been suggested which ensures Safety requirements are met. Verification starts at the initial phase of development in devising a Composable and Compatible model reducing safety faults and hence reducing the cost of development. Verification assists Validation in fault reduction reducing the tolerance towards safety negligence.

8. FUTURE WORK

One of the problems that arise in safety system is the changes and updation in the system and the adaptation to new external environment. The dependence between components changes and hence its compatibility. Hence the verification and validation of safety properties poses a challenge driving more research.

REFERENCES

[1].Peter G. Neumann (2004). 'Principled Assuredly Trustworthy Composable Architectures' (Report).

[2]. Zhe Dang, Oscar H. Ibarra, Jianwen Su.Composability of Infinite-State Activity Automata.Algorithms and computational Lectures notes in computer science, Volume 3341,2005, pp 377-388

[3]. Damiano Angeletti , Enrico Giunchiglia ,Massimo Narizzano , Alessandra Puddu ,Salvatore Sabina. Using Bounded Model Checking for Coverage Analysis of Safety-Critical Software in an Industrial Setting. Springer Journal, 2010,45:397–414.

[4]. Linear Temporal Logic.http:// www.voronkov.com/lics_Chapter14.pdf

[5] FormalVerification:Computation TreeLogic,http://www.inf.unibz.it/~artale/FM /slide4.pdf

[6] V. V. Lipaev. A Methodology of Verification and Testing of Large Software Systems Programming and Computer Software, Vol. 29, No. 6, 2003, pp. 298–309.

[7] De Alfaro, L., Henzinger, T. A., & Stroelinga, M., Timed interfaces. Proceedings of the Second International Conference on Embedded Software, (EMSOFT 2002), 2491, LNCS. Springer.2002, pp. 108–122.

[8] Robinson-Mallett, C., Hierons, R. M., & Liggesmeyer, P.,Achieving Communication Coverage Criteria in Testing, Workshop on Advances in Model-Based Testing,Raleigh, NC,2006

[9] Clarke, E. M., & Emerson, E. A. Design and synthesis of synchronization skeletons using branching time temporal logic. Proceedings of the Workshop on Logic of Programs. Yorktown Heights, NY,LNCS 131, Springer Press,1981,pp. 52–71.

[10] Clarke EM, Emerson EA, Sistla AP. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans Program Languages Syst ;8(2),1986,pp:244–63.

[11] Gluch PD, Brockway J. An introduction to software engineering practices using model-based verification. CMU/SEI-99-TR-005, 1999.

[12] Pietro Braione , Giovanni Denaro •,Andrea Mattavelli ,Mattia Vivanti •,Ali Muhammad,” Software testing with code-based test generators: data and lessons learned from a case study with an industrial software component” Software Qual J,Springer(2013), DOI 10.1007/s11219-013-9207-1

[13] J. Lahtinen , J.Valkonen , K.Bjorkman , J.Frits , I.Niemela , K.Heljanko , Model checking of safety-critical software in the nuclear engineering domain, Reliability Engineering and System Safety 105 (2012) ,pp 104–113