Turkish Journal of Computer and Mathematics Education *Vol.12 No.10(2021), 1313 - 1319*

Research Article

A Cellular Automata based Safe Regression Test Selection Approach for Soa Based Applications

Rajanikanta Mohanty[a], Binod Kumar Pattanayak[b], Satya Sundara Mohapatra[c]

[a] Department of CSE, PSIT, Kanpur, India, Email: rkm.bbs@gmail.com
[b] Department of CSE, ITER, Siksha o Anushandhan Deemed to be University, Bhubaneswar, India, Email: binodpattanayak@soa.ac.in
[c] Department ofIT, PSIT, Kanpur, India. satyasundara123@gmail.com

Article History Received: 10 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 28 April 2021

Abstract: Service Oriented Architecture (SOA) based applications mostly comprise of heterogeneous as well as self-contained independent web services. Such applications need to be modified in order to fix the errors or with an intention to improve their underlying functionality. Incorporated such modifications are supposed to be quick and need to be supported with instant verification. Regression Test (RT) is necessary in order to guarantee the modifications should not consequently lead to any adverse effects. Regression Test Selection (RTS), considered to be one of the cheapest strategies, is intended at reducing the cost of implementation of RT. In this paper, the authors present a cellular automata based approach that enables the respective professionals for application of a safe RTS technique to SOA based applications or services in an end-to-end fashion. Safe RTS technique ensures that no modification specifying the tests should be unselected. A simplified navigational subsystem is used in this paper in order to elicit the proposed approach that incorporates 3 different services.

Keywords: SOA Based application, Regression Testing, cellular automata, RTS.

1. Introduction

Web services, a class of SOA based applications, have become a common mechanism that usefully supports inter as well as intra enterprise business processes. As the business processes tend to change quite often, the SOA based applications undergo rapid and frequent changes too. Hence, the modifications must be substantiated by rapid verification. In this context, in order to ensure that the incorporated modifications must not adversely affect the functionality of an existing system, one needs to take into consideration the test cases that were used previously to test the existing system prior to incorporating the necessary modifications again into the existing system. This process of testing again is referred to as regression testing. The key idea here is to use RTS, whose objective is to reduce the size of the function that is to be retested. RTS plays an important role in SOA based environment in which numerous services or service based applications are interacting with each other as retesting of all the previous test cases would certainly be presumably extremelycost-consuming. The necessity of RTS was well established for the traditional software system [1]. Nonetheless, there does not exist any proper mechanism available to directly use safe RTS to SOA based applications or independent services. Safe RTS techniques necessarily assume white box, that is, code based testing. Code based testing however cannot be applied directly to the service interactions that comprise of services that are implemented in various languages or service providers those who do not share their source codes. This paper focuses at safe RTS technique for SOA based applications and services. As a consequence of the inherent autonomy of the SOA based applications, a newer set of challenges do arise before the service providers as the modifications can occur at any point of time. We present here a Grey-box approach that supports a safe RTS technique for verification of services in an end-to-end manner. Our approach is presumably on the basis of the safe RTS algorithm as proposed by Rothermel and Harrold [2].

1.1 Cellular Automata

The evolution of Cellular automata has been proven to be quite efficient for conducting arbitrary information processing. A significant application is contained in the theory as well as practice of finding a method for unification of the information processing. In explicit cases one can formulate cellular automaton rules which will permit specific computations to be carried out. Challenges lies in identifying the structures in a cellular automaton which is capable of interacting in order to mimic the components of conventional compute systems. But all these approaches are strongly based on analogues with conventional serial-processing computers. However, information processing in a cellular automaton occurs is necessarily distributed as well as parallel fashion, and it is essential to work on a new framework to apply such technique. Such a framework would helpful in analyzing many physical systems wherein the information processing is supposed to be distributed.

Further, Cellular automata approach is supposed to be a statistical approach. It assumes devising as well as describing the attractorsthat are needed for the global evolution of cellular automata. All the traditional configurations contained in a particular basin of attraction may be considered to be some instances of a specific

pattern for the reason that their evolution toward the attractor can be considered to be recognition of the pattern. This approach is supposed to be necessarily effective in a case where the basins of attraction are supposed to be local in terms of the space as it is in the case of image processing. But, devising the attractors for more generalized problems appears to be considerably difficult. An attempt can be made in this direction by virtue of considering the basins of attraction as sets of sequences corresponding to particular formal languages.

Another probable approach can be to use the symbolic representations for various attributes or components of cellular automaton configurations. But here, the structures that are used in traditional programming languages may appear to be largely inappropriate. The underlying organization of computer memory words into named locations, stacks and so on, is not appropriate for cellular automata wherein the processing elements can not be distinguished from memory elements. Rather, it can be assumed that the data could be represented in the form of an object such as a graph wherein the necessary transformations can be conducted in parallel. This chapter covers the basic literature on cellular automata, related definitions and notations thereby focusing on its applications in information processing.

The term *cellular automata* is plural. Our code examples will simulate just one—a *cellular automaton*, singular. A cellular automaton is a model of a system of "cell" objects with the following characteristics.

- The cells live on a *grid*.
- Each cell is identified by a *state*. The number of states is finite. The simplest example is that a state has the two possible values as 1 and 0 (otherwise referred to as "on" and "off" or "alive" and "dead").
- Each cell has a *neighborhood*. This can be defined as a list of adjacent cells.

2. Background

A test case in software testing always represents a set of given inputs to a program P. A test suite thus can be defined to be a set of test cases where a test-run refers to the execution of program P with reference to some specific test suite T. After modification of program P that results in P′, if the same tests are run on P′ from T again, we thus perform the regression testing. As elaborated previously, RTS techniques mostly tend to reduce the incurred cost of RT before selecting the test suite T′ that is a subset of T, and using the test suite T′ for testing program P′. In case, the original test suite is found to be comparatively more expensive in order for running the reduced test suite with the implementation of the RTS technique, the reduction in cost can be acquired. We are using CFG from the source code as the background of our RTS technique. CFG represents a directed graph wherein each node refers to a code entity and an edge depicts the transfer of the flow of control from one code entity to the other. The entities here are considered to be statements, classes, methods or components. In addition to it, the CFG is supplemented with the coverage information relating to the test cases while testing a particular code entity and inversely, the code entities tested under each of the test cases. The comparison of P with P′ is done by the RTS algorithm to find out the dangerous edges. The dangerous edges here refer to the code entities that presumably tend to behave in a different way under one test case resulting from the differences existing between the programs P and P′. This approach presumably ensures that any test case which does not cater to a dangerous entity, will presumably behave in the same fashion in program P as in program P′, and thus, must not give rise to a new defect in program P'. Hence, it is safer to choose only those test cases whose coverage maps to one of the dangerous edges or even more. This is the procedure where the safe RTS technique minimizes the number of test cases thereby maintaining the same level of confidence as as could occur in the case of retest-all approach. Our CFG approach conducts mainly three steps: (1) it generates a CFG for the program P′; (2) it identifies dangerous edges by means of comparing CFGs of P and P′ using dual-traversing across the two CFGs; (3) it selects those tests that need to be re-run, from the test suite, based on the available thereby coverage information pertaining to the dangerous edges.

We are using state diagram representation of cellular automata from the source code as the background of our RTS technique. State diagram represents each node having a code entity and connectivity between states refers to the flow of control from one code entity to the other. The entities as mentioned earlier, can be statements, classes, methods or components. In addition to it, the state diagrams are supplemented with coverage information about test cases while testing a given entity and inversely, the entities tested by each of the test cases. The comparison of P with P′ is done by the RTS algorithm to find out the dangerous edges. The dangerous edges here refer to the program entities that may behave in a different way under a single test case due to the differences existing between P and P′. This technique ensures that any test case which does not cover a dangerous entity, will behave in the same way in both P as well as P′, and thus cannot necessarily lead to a new fault in P'. Thus, it is safer to choose the only test cases the coverage of which maps to one or several dangerous edges. This is the procedure wherein the safe RTS technique tends to minimize the number of test cases thereby maintaining the same level of confidence as provided by the retest-all approach.

The technique elaborated in this work uses cellular automata based State diagram to demonstrate on performing three main steps: (1) it demonstrates the flow of control for P′; (2) it determines the dangerous edges by virtue of comparison of the state diagrams of P and P′ using dual-traversing across the two state diagrams; (3) it selects those tests that need to be re-run, from the test suite, on the basis of the available coverage information of dangerous edges.

3. Related Work

The spectrum of research work conducted by authors on regression testing of SOA based applications is significant. Authors in [3] propose a visual framework for monitoring and controlling a network of home appliances, where ten home applications from the distributed service components of the monitoring system using Flash and Java Servlet. A safe regression testing algorithm is detailed in [4], where authors focus on three different situations: 1) connecting to a newly established composed web service; 2) addition or removal of an operation pertaining to the respective composed web service and 3) modification of the specification of the web service in order for application of this algorithm. Authors in[5] present an implementation of automated session data repair for web application regression testing. A novel safe regression testing technique for various web applications is proposed by authors in [6] that considers changed elements along with the other potentially affected ones. An automated regression test suite for web services is addressed in [7] that uses parsing of WSDL file and then generating SOAP requests along with the required values of the requests, which can be searched in the data base. Authors in [8] detail the implementation of the synchronization of multi-window requests for regression testing of web services at the server side that makes it simpler to specify values that need verification during development of web applications. In order to verify that a system did not regress after some parts of it are modified during maintenance phase of web services, an approach for model-based regression testing is proposed in [9]. Authors in [10] propose a regression testing approach especially meant for composite web services that helps in finding the defect in composite web services that makes test data and testing behavior independent of each other. Li Chen Ziyuan et al. [11] propose a test case prioritization technique for the regression testing of web services by analyzing the dependence relationship by virtue of using the information relating to the control flow and data flow thereby constructing a weighted graph and performing impact analysis for identification of modification-affected elements. Authors in [12] present a hierarchical regression test selection algorithm for SOA based applications and claim to have implemented it in a SOA environment resulting in better results as compared to the existing approaches.

4. Construction of Cellular Automata based State Diagram

This is the first step in performing safe RTS technique. For each service, we construct a state diagram for each operation. Every state diagram here is practically identified by the name of the corresponding operation along with the Uniform Resource ID (URI) of the service. Here, the issue of the granularity level of each state matters significantly. In addition to cost precision trade-off, there may be developers unwilling to share the source code and design documents. These are also the reasons related to the issue of different granularity levels.

Simple services' state diagrams can be generated like that of monolithic applications, since all of them needed the artifacts are available during generation of state diagrams. Explaining the procedure using state diagram for composite services assumes practically the same pattern like the simple services except the fact that composite services have call nodes. Here, we restrict our method only to the static service composition. A call node holds the URI as well as the name of the method relating to the operation that is being invoked. Here, the Call nodes are entered into the state diagram as per the previously computed patterns. We determine the service that is invoked along with the operation on it using the code, by applying a pattern based approach, and enter that call node into the state diagram with directional edge at the proper location within the entire state diagram.

The state diagram with directional connectivity can demonstrate the functionality of call nodes. Thus with direction oriented connectivity, state diagram helps in classifying those process which are not ready for use in the RTS processing, then we may call it as a non-terminal state diagram. All state diagrams constructed for simple services may be called as terminal state diagram time being for explaining the control flow of our problem. So, the end part of this process is responsible for transformation of the non-terminal state diagram to terminal state diagram. Transformation of non-terminal state to terminal state is straight forward and can be achieved by minimizing the steps or ignoring non performing states and making the completion of the process. Replacing all call nodes with directed connectivity of state diagram of the services being called can be studied theoretically. We call the desired service thereby asking for its terminal state using the URI along with the method name. As we ask for only terminal states, we avoid wasteful computations and communications. After receiving the newly obtained terminal state, we then enter it into the state diagram in place of the state called by the directed node. Henceforth, we will mention such directed transition as call node. The predecessors of all the call nodes will

receive their respective first nodes for the newly achieved states as their successor nodes, and here, the end nodes of states get the successor nodes consequently, i.e., the call nodes as their successor nodes. After the completion of this process, for each of the call nodes, the specific state now becomes a terminal state that is ready for use in the process of RTS processing. This process using CFG is detailed in [13]. In recent times, services run with some frameworks that may be treated as libraries. Hence, the codes provided in the frameworks are unchanged and can be omitted in our consideration. Thus, each operational state in each of the services will be generated from the only functions that factually do implement the operation. In this proposed approach, every called operation is predetermined by the program without involving service directory or look-up. Regression testing is supposed to perform perfectly only in case the system is supposed to be deterministic during the procedure of testing. In addition, our approach can be applied to static service composition only. The hash-code field here is used for identifying the modifications in the code. In order for recognition of the modified code, we need not know the actual code. Hashing is referred to as a one way encryption mechanism which necessarily supports comparison of equality and that is considered to be enough for discovering changes thereby protecting the intellectual property. Hence, each of the nodes (or states) in the state diagram holds the hash-code of the respective program entity. If the level of granularity within a node refers to the design-level, then the hash-code of it is assumed to be null.

Recognizing Dangerous Edges or connectivity in-between the states

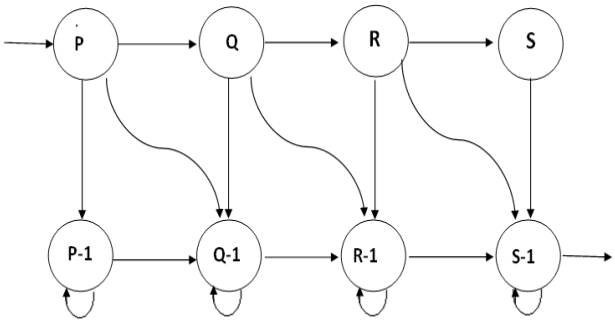
The subsequent step of the safe RTS technique continues after the modification of the code of the identified operation [1]. Actually, a new state diagram is constructed and then, it is compared with the old state diagram by virtue of a dual-traversal across both the state diagrams, consequently leading to identification of the parts of the directed graph that are different. Such differences are either structural or as per the contents of the respective states. Since dual-traversal is performed specifically for structural differences, detection of such differences becomes straightforward. The hash code of each state of the state diagram is compared in order to detect the differences in the contents. After the identification of the changed nodes, the downstream edges can be marked as dangerous edges. Three specific cases should be taken into consideration for application of this approach to the distributed web services: (1) the respective old state diagram must be with a granularity level that is different from the earlier one; (2) the modification is performed from a remote site; (3) the allowable level of granularity must be coarser at the remote site as compared to that of the new state diagram.

Case (1) is intended for modification of a matured component. The detailed state diagram locally is created by initiating the RTS procedure. The newly created state diagram is then inserted in place of the old abstract state in the old state diagram, and thus, the new terminal states are obtained. In this process, we generate the new state diagram locally and then insert it into the old state diagram at the same time. Since time is a factor, instead of traditional automata we have considered cellular automate. This is intended in order for dual-traversal comparison between the old and new transition state in the state diagrams. Otherwise we would require to mark every changed state in the new state diagram as either “changed” or with some other notation.

Case (2) results from the distributed nature of the traditional web services. Modification of one operation O, conducted in one service consequently affects operations of other services which call O directly or indirectly. We use a call graph in order to identify such potentially impacted remote operations, where each node refers to an operation of a client program that consequently refers to a call relationship. Here, the operation that is represented by the source node calls the operation that is represented by the pointed node. In the call graph, each node is marked with a box pointing to a web service. However these “service boxes” do not play any role in the graph computation, rather only point to the locality of the associated operation. Thus, if an operation A calls operation B, then any changes incurred in B shall consequently affect A. Let us consider anode N associated with an operation n. As a rule, starting with N, all of the reversely reachable here nodes comprise of the inverse closure of N in the call graph. Hence, all the operations associated with the nodes belonging to the inverse closure of N are affected by the change in n and need to be tested. For this reason, all the services implementing these operations need to perform one complete cycle of the RTS procedure. A detailed elaboration of inter-procedural control dependence is included in [14]. In comparison, our approach is significantly simplified as we mostly focus at distributed and autonomous services.

Case (3) is technically unreasonable. In fact, trust/ownership gap between the partners may exclude the scenario of one service provider providing a statement-level and can be demonstrated using transition states in the state diagram to another, after a modification is incorporated into an operation. Thus, the owner of the modified operation will implement the allowable level of granularity in the new transition of the state diagram using a best-effort approach, leading to another state diagram. It means the transition status representing method in the new state diagram can be accepted as accurate ones. But, if the operation level is the only allowable granularity level, then the representing state needs to be marked as “changed”.

Fig.1 depicts a state diagram with directed connectivity that refers to a scenario, where two changes take place during the same transition. In this scenario, one changed operation or transition can be reachable from the other changed operation. In fact, these two changes arrive in a conflict. For example, in Fig.1,if S-1 and Q-1 are changed, then a failure in the regression test conducted by P cannot spot the location of the fault provided the P mustn't be in the intermediate result. So we need to initialize the P first. If Q-1 changes first, then P would select test cases as per the changes in Q-1 and then test Q-1. Thus, if testing Q-1 is successful, then changes in Q-1 can be realized. Here the Q is a state which a product obtained from P after getting some input. So if P is change then directly Q is also alter. Hence we need to make the Q as fixed in accordance with occurrences of Q-1. Otherwise, afterwards if R tests S-1 and consequently fails, then it refers to previous test for Q-1 being wasteful as Q-1 might be using an unusable operation R-1. In such a case, both S-1 and Q-1 should rollback. To overcome such cascaded rollback, changes must be incorporated in the directed transition in "initial transitional state changes first" manner. In case of a failure, intermediate states changes must not be incorporated before the state that is having faulty unit is spotted and fixed.



State Diagram with directed Connectivity

5. Case Study: Navigational Subsystem

We apply our approach to a simplified Navigational Subsystem that has been changed as mentioned in the example in Section 4.2. This subsystem comprises of three services and service based application. These three services are SR1, SR2 and SR3. SR1 is the "Destination Selection Service" that selects the destination from the devices like PDA, Mobile phone or Automobile navigator. SR2 is the "Route Calculation Service" that calculates the route by surface transport. SR3 accepts the request and forwards the received thereby incoming request to the appropriate service with respect to the type of request. The application called AP1 is a simple proxy that takes the request and outputs the result in a service. Their pseudo codes are detailed below.

pseudo code

SR1 (Destination Selection Service)

1. request (a service place)

{

2. if (place is found)

{

3. if (place is in the map)

{

4. take it as destination

5.return successful;

}

6. else return error

}

7. else print ("place not found")

}

SR2 (Route Calculation Service)

```
1. request (service route)
{
2. if (route exists and route is in the map)
{
3  Compute the route
4. return successful;
}
5. else  print("route not found")
}
SR3 (Activating Service)
1. request (service activation)
{
2. if request is (destination calculate)
{
3. return call SR1;
}
4.else if          (request is route calculation)
{
5.return call SR2;
}
6.      else return print ("no route found")
}
AP1 (application1)
request (service/place/route)
{
if (parse (service) is successful) {
Print call SR3;
}
else
Print error ("service not available")
}
```

6. Conclusion and Future Work

Now-a-days it has become very important to ensure the quality of SOA based applications moving across loosely coupled system. With an aim to improve the testing system, more test cases are added. This increases the cost significantly during the process of RT. Hence, safe RTS techniques are becoming a necessity for reduction of the testing time for each of the carried out modifications. In this paper, we have presented a procedure to use a safe RTS technique to SOA based application. Our next aim is to develop a framework using model based testing methodology that can support this scheme in a larger dimension.

References

1. M.Bruno et al, "Using Test Cases as Contract to Ensure Service Compliance across Releases", Proceedings of 3rd International Conference of Service Oriented Computing

2. (INCS 3826), pp. 87-100, 2005.
3. G. Rothermel and M. J. Harold, "A Safe, Efficient Regression Test Selection technique", ACM Transactions on Software Engineering and Methodology, Vol.06, No.02, pp.173-
4. 210, April 1997.
5. T. Kimura, H. Tamada, H. Igaki, M. Nakamura and K. Matsumoto, "A Visual Framework for Monitoring and Controlling Distributed Service Components", IPSA SIG Technical Reports, Vol. 2005, No.60, pp. 245-250, 2005.
    A. Tarhini, H. Fouchal and N. Mansour, Regression Testing Web Service-based Applications, Proceedings of IEEE/ACM International Conference on Compuetr Systems and Applications (AICCSA2006), UAE, 2006.
6. N. Alshahwan and M. Harman, "Automated Session Data Repair for Web Application
7. Regression Testing", Proceedings of the International Conference on Software Testing,
8. Verification and Validation, pp., 2008.
    A. Tarhini, Z. Ismail and N. Mansour, "Regression Testing Web Application", Proceedings of the International Conference on Advanced Computer Theory and Engineering, pp. 902-906, 2008.
9. Kumar S, G.P. Kumar and A. Dhawan, "Automated Regression test Suite for testing Web Services", Proceedings of the International Conference on Advances in Recent Technologies in Communication and Engineering, pp. 590-592, 2009.
10. T. Shimomura, K. Ikeda and M. Takahashi, "Synchronization of Multi-window Requests for Server-side Regression Test of Web Applications", Proceedings of Ninth International Conference on Quality Software, pp. 129-134, 2009.
11. T.A. Khan and R. Heckel, "A Methodology for Model-Based Regression testing of Web Services", Proceedings of 2009 Testing: Academic and Industrial Conference - Practice and Research Techniques, pp. 123-124, 2009.
12. Yang, J. Wu, C. Liu and L. Xu, "A Regression Testing Method for Composite Web Service", Proceedings of the 2010 International Conference Biomedical Engineering and Computer Science, pp. 1-4, Penang, 2010.
13. L.C. Ziyuan, W.L. Xu and L.B. Xu, "Test case Prioritization for Web Service Regression Testing", Proceedings of Fifth International Symposium on Service Oriented System Engineering, pp., 2012.
14. K. Rana, H. Shah and C. Kapadia, Regression Testing of Service Oriented Software, International Journal of Innovative Technology and Exploring Engineering, Vol.8, No.10, pp.448-453, 2019.
15. M.Ruth and S. Tu, "A CFG-Based Regression Test Selection for Web Services", Proceedings of Second International Conference on Internet and Web Application and Services (ICIW'07), pp. 01-06, 2007.
16. S. Sinha, M. J. Harold and G.Rothermel, "Inter-procedural Control Dependence", ACM Transaction on Software Engineering and Methodology, Vol.10, No.01, pp.209-254, April,2001.