Extenics-based Test Case Generation for UML Activity Diagram

Liping Li\(^a\), Xingsen Li\(^b\), Tao He\(^c\), Jie Xiong\(^d\)

\(^a\)Computer and Information Institute, Shanghai Second Polytechnic University, Shanghai, 201209, China
\(^b\)Management School, Ningbo Institute of Technology, Zhejiang University, Ningbo, 315100, China
\(^c\)Software Engineering Department, Shenzhen Institute of Information Technology, Shenzhen, China
\(^d\)College of Information Science & Technology, University of Nebraska at Omaha, Omaha, NE 68182, USA

Abstract

Extenics is a new discipline for modeling contradiction problems with formalized methods and transformation. This paper for the first time combines software testing with Extenics and proposes an automatic approach to generating test cases from UML activity diagrams based on Extension Theory (Extenics). In order to find more defects in software system by minimized test cases, we design algorithm to construct the Euler circuit and generates test sequences automatically by Euler circuit algorithm. Our preliminary result shows that test cases generated not only satisfy the specified test coverage criteria but also the number of test cases is decreased.

© 2013 The Authors. Published by Elsevier B.V.
Selection and peer-review under responsibility of the organizers of the 2013 International Conference on Information Technology and Quantitative Management

Keywords: Testing; Test cases; UML activity diagram; Extenics; Euler circuit

1. Introduction

Testing is one of the most frequently used approaches to verify the quality of software. Automated software testing is imperative because manual testing takes much time and costs and is error prone. A test case is a sequence of conditions or variables that satisfied with certain test coverage criteria. With the help of test cases, test engineers will find whether a software system is implemented according to the requirements. Test cases can be generated automatically from a formal system model. And generating test cases from a system model often finds inconsistencies and ambiguities in the system requirement and design.

Extenics is a new discipline for dealing with contradictory problems with formalized methods and transformation [1]. It considers the matter, the characteristic and their measures together with their relation and their change. In realistic world, systems usually contain a great deal of information, testing all its possible paths are impractical. In order to reduce the testing cost and find more errors, researchers have studied how to cover as many as possible paths in the system by the minimized test cases. This paper is trying to solve this contradictive problem using extenics theory.

UML activity diagram is a very important and visualized diagram in UML. It describes the sequential
or concurrent activities among a group of objects. It can be used for business modeling, control and object flow modeling, operation modeling etc. The main advantage of the activity diagram is its simplicity and ease to understand the logic flow of the system. In recent years, test cases generation from UML activity diagrams has gained some attention [2-5]. D Kundu et al. [2] presented an approach to generating test cases from activity diagrams based on UML 2.0 and with use case level. They used the activity path coverage criterion to find the synchronization faults and faults in loops. Wang Linzhang et al. [3] proposed a systematical approach for test cases generation from UML activity diagrams using gray-box testing method. Test scenarios satisfying specific coverage criterion were generated from the activity diagrams. They used basic path coverage to avoid path explosion in the loop. Chen Mingsong et al. [4] present an approach to generate reduced test suite for an implementation using activity diagrams. They first randomly generate abundant random test cases. Then, the program execution traces were obtained by running the program with these test cases. Last, by comparing these traces with the activity diagram according to the simple paths coverage criteria, they got a reduced test suite that meets the test adequacy criteria. Chris [5] proposed an approach for test generation from Formal Activity diagram (FAD). They formalized the Activity diagram by extending the Activity diagram’s syntax and semantics.

This work is different from all above works. We present an approach to generating reduced test cases from UML activity diagrams partly based on Extenics. First, we model activity diagrams by Rational Rose to represent system requirements. Second, design algorithms to extract the key information of activity diagram by analyzing the .mdl file of the system. Third, we visualize the refined .mdl file to a directed graph through Graphviz [7]. In order to find more defects by minimized test cases, we transform the directed graph to the Euler circuit based on Extenics transformation method. Finally, test cases were generated automatically using Euler circuit algorithm. A typical example ATM system is used to illustrate this approach. A corresponding tool prototype has been implemented. Result shows, this approach can generate test cases automatically and reduces the number of test cases.

The remainder of this paper is organized as follows: Section 2 briefly introduces the theory of Extenics and UML activity diagram using the ATM example. Section 3 presents our approach to draw the main information of activity diagram. Section 4 describes our algorithm to convert activity diagram into the Euler circuit and generate test cases automatically for activity diagram using the Euler circuit algorithm. Section 5 concludes the paper and highlights our future work.

2. The Preliminaries

2.1. Extension Theory-- Extenics

In 1983, the article Extension Set and Non-compatible Problems was published in Journal of Science Exploration, which proclaimed the birth of the new discipline—Extenics. Its objects of study are contradictory problems in the realistic world [1, 9]. It is a new discipline for dealing with contradiction problems with formalizing model and makes novel decisions [10]. During the past 20 years, a series of particular extension methods have been developed, such as matter-element extension method, matter-element transformation method etc. The combination of extension methods with other engineering fields generates extension engineering methods to solve contradictory problems in various fields, such as information, automation, design and management fields [9-13].

Extenics constitutes of element theory, extension methodology and extension engineering. A matter has many characteristics. From the view point of material nature, every matter is the entity of the imaginary part and the real part. Similarly, each element has four extensibilities: the divergent nature, the conjugate nature, the correlative nature and the implicative nature. As to the conjugate nature, there are four pairs of conjugate parts including the real part and the imaginary part, the soft part (relation structure between parts of a system) and the hard part (each part in a system), the latent part (unnoticeable element or forthcoming change) and the apparent part (noticeable element), the negative part (the part creating positive value to the goal) and the positive part (the part creating negative value to the goal). A matter can be expressed as an ordered 3-tuple

\[ R= (N, c, v) \]  

where \( R \) is the matter-element, \( N \) represents the matter, \( c \) is the characteristics, and \( v \) is the \( N \)'s measure about the characteristics \( c \), i.e. \( v = e(N) \).

Up to now, Extenics establishes matter-element \( R=(N,c,v) \), affair-element \( I=(d,b,u) \) and relation-element \( Q=(s,a,w) \) (each has attributes/characteristics and their measure) to describe matter affair and relation. Matter-
element, affair-element and relation-element are collectively called basic element. A matter can has many characteristic-elements, which can be described by n-dimensional matter-elements:

\[
M = \left[ \begin{array}{ccc}
N_{m1} & c_{m1} & v_{m1} \\
N_{m2} & c_{m2} & v_{m2} \\
\vdots & \vdots & \vdots \\
N_{mN} & c_{mN} & v_{mN}
\end{array} \right] = (N, C, V) \quad (2)
\]

All information and knowledge related to the goals and restrain conditions in software testing can be expressed as matter-elements.

Extenics also present transformation methods. By certain transformation, one thing that doesn't have property \( P \) can be turned into another thing that has the property \( P \) [11]. Extension set can describe this kind of transformation quantitatively.

2.2. UML activity diagram

UML activity diagram include activity states, transitions, decisions, swimlanes, forks, joins, objects, signal senders and receivers. Activity states and action states are denoted with round cornered boxes. The edges represent the occurring sequence of activities, control flows, message flows and signal flows etc. Because UML is a semi-formal language and corresponding semantics are lacked for test case generation. We formalize the activity diagram using the n-dimensional matter-elements \( AC=(N, Cm, Vm) \). \( N \) represents the activity diagram; \( c_{mi} \) is the characteristics of the activity diagram, such as the activity states, transitions, guard conditions, initial activity state and final activity state. \( v_{mi} \) is the \( N \)'s measure about the characteristics \( c \). The activities are represented as affair-element \( I=(d, b, u) \); Transitions between activities are represented as relationship-element \( Q=(s, a, w) \).

Figure 1 is the activity diagram for ATM example. The activity states are rounded rectangles such as “Insert card”, “Enter password” and “authorize” etc. There are three swimlanes: customer, ATM and bank. Diamonds represent decisions, bars represent the fork or join of concurrent activities, start is the initial state and end is the final state.

![Fig.1. ATM Example](image)

3. Refine the activity diagram

We first use Ration Rose to draw the activity diagram. But the .mdl file is more than complex and redundant for read and test cases generation. So, we refine the activity diagram. When we analyze the file, we found a very simple rule: all relevant information of the activity diagram is included in the directory of “(object Active”, and organizes in the “( )” block. Therefore, our algorithm search from the beginning of the file, if the symbol ”(object Active“ and...
other useful keywords are found in a row, our system reads the necessary content and write them into a file according to the left and right parentheses match method. Actually, we use a dynamic matching extraction method to draw the useful information, and stored in strings. Then write these strings to a file in accordance with state unit and transition unit. Figure 2 is the refined XML file for ATM example.

4. Transform the activity diagram to the Euler circuit

The tool for solving contradictory problems is extension transformation. Through certain extension transformations, unknowable problem can be transformed to knowable problem, unfeasible problem can be transformed to feasible problem, false proposition can be transformed to true proposition, and wrong inference can be transformed to correct inference. These transformations are the commonly referred ideas, tips and methods. There exist many contradiction problems in software testing, which one of them is using the minimized test cases to find the most faults in the system. In order to reduce the testing cost, many studies about test cases reduction have been studied. Some studies [6] showed that the size of test suites can be dramatically reduced and the fault detection of the reduced test suites is adversely affected. But some [7] found no distinct effect in fault finding ability between the full suites and the reduced suites. Heimdahl et al. [6] studied the effect of test suite reduction by seeding representative faults in Flight Guidance system example. Their results showed that one could dramatically reduce the automatically generated conformance test suites while maintaining desired coverage, but the fault finding of the reduced test suites was adversely affected. Wong [7] discussed the problem of the effect on fault detection of reducing the size of a test set while holding coverage constant. Their result showed that the decrease in fault detection capability for the minimized test sets is slight even when the reduction in test set size is great.

In this paper, we transform the UML activity diagram to the Euler circuit for the Euler path is a trail which visits every edge exactly once. This characteristic of the Euler circuit can ensure the transition coverage and optimal the test cases.

4.1. Construct Euler circuit

Euler circuit is one of the most age-old graph problems in graph theory. A circuit is a path that starts and ends at the same node (vertex). Euler circuit is a circuit of graph $G(V, E)$ if it is exactly traversal each edge $e \in E$ in $G$ only once. This paper proposes a method to construct the Euler circuit from directed graph and generating test cases using Euler circuit algorithm. We first transform the UML activity diagram to the directed graph and then to construct Euler circuit in order to satisfy the transition coverage criteria and make the number of generating test cases is minimized.

We use class to store each node and keep the outdegree and indegree of each node. Take advantage of Euler circuit character, we can generate test cases as minimum as possible. The methods that transforming directed graph to Euler circuit is shown as below:

1) Add some auxiliary edge to make all end nodes in the directed graph to point to the start node, to form loops.
The sufficient and necessary condition of Euler circuit is that the graph is a strong connection graph and the indegree of all nodes is equal to the outdegree. Clearly, all nodes are connected if the end nodes point to the starting node. So we only should solve the problem of nodes indegree and outdegree.

2) Let $\text{num}_i$ is the value of indegree subtract outdegree of node $i$, i.e., $\text{num}_i = \text{indegree}_i - \text{outdegree}_i$. Access every node, if $\text{num}_i > 0$, then we find the node which $\text{num}_i$ is minimized among all nodes leave from node $x$ and add auxiliary edges from node $x$ to the node which $\text{num}_i$ is minimized. If $\text{num}_i < 0$, then we find the node which the value of $\text{num}_i$ is maximum among all the nodes point at node $x$ and add auxiliary edges from node $x$ to that node. Repeat the above steps until the $\text{num}_i$ of all nodes in directed graph equal to 0. After several times to add the auxiliary parallel edges, we can make the directed graph to become the Euler circuit.

![Fig.3. Add some auxiliary edges](image)

For example, in figure 3 (a), the $\text{num}_x$ of node X satisfy the condition $\text{num}_x > 0$, so we should find the node which $\text{num}_i$ is minimized among all nodes leave from node $x$, is node i. One auxiliary edge (red one) is added for node i. In (b), the $\text{num}_x = -1$, so we find the node which the value of $\text{num}_i$ is maximum among all the nodes point at node $x$ and add one auxiliary parallel edge (red one) from j to x. As we can see, the $\text{num}_i$ of all nodes is 0 after adding edges.

3) There may exist loops after step 2. But loops are excrescent because they do not affect the outdegree and indegree of nodes. So we must delete the loops. The method to remove loops will be introduced in chapter 4.2. After step 3, the graph can be covered by one path.

4.2. Generate test cases base on Euler circuit

An Euler circuit exists if it is possible to travel over every edge of a graph exactly once and return to the start node. After the step of adding auxiliary edges, the graph will form some loops, but loop does not affect the degree of node, is redundant. So, we must detect and remove the loops.

The algorithm to remove loops of the directed graph is shown as below:

Put the transitions artificially into a two-dimensional array, a transition is added $n$ times will be recorded number $n$. Select a node randomly as a start node, transit to other nodes and push them into the stack. The traversal number of transition will decrease 1 when traversing once. When the transition number of a node equals to zero, we will backtrack to the previous node to find a new way and this node is popped out of the stack until come to the start node. If there is only one node in the stack, then this node does not belong to the loop; if more than one node, that means we have to found the loop and remove it. Corresponding, the number of transition will minus 1 until there has no loop any more. After processing all nodes, we can get the paths without loop.

The graph can be traversal by one path after the step (3) in chapter 4.1. At this time, we should find the path which traverse each transition only once from state node using the Euler circuit algorithm.

The algorithm for one path to cover the Euler circuit is shown as below.

The basic form of the Euler circuit is a loop interwoven, so randomly select a path which starts with the start node; the end of this path is inevitably the start node, so there will be the first loop. And then find if there exist nodes which have extra degrees on this loop. For example, node i have 2 indegree and 2 outdegree, after travel once there will has surplus. Then we start from the node which has redundant degree, i.e., the node which has some edges which have not been covered, find a new loop embedded in the original loop. Repeat until you find the path which all nodes have no extra degrees. This path is the path which covers every edge of the directed graph exactly once.
Output the path from the start node is the test paths satisfy the transition coverage.

The test cases generated automatically for ATM example satisfies the transition coverage is shown as figure 4.

Our approach for generating test cases satisfies activity state coverage is transferring it to transition coverage. Because this can ensure the number of test cases is reduced. And we also can use Euler circuit to generate test cases. But we should delete all the redundant transitions in order to reduce the number of test cases.

The algorithm for generating test cases satisfies state coverage is composed of the following steps:

1. First copy the original directed graph, then delete all the shortcuts of graph. For example, if there exists a path from A to B, i.e., A->B, but there are also exist another path A->C->D->B, then we delete A->B.
2. Delete all redundant transitions. For example, there exists a transition A->B, if there has a path from start node to B without covering the transition A->B and A also can move to the end node without covering the transition A->B, then we delete A->B.
3. After the two steps, we add the auxiliary edge to make all end nodes to point to the start node. And record the indegree and outdegree of all nodes. In order to make the degree of all nodes equal to 0, we add the parallel edges to original graph, just like the step 2 of transition coverage.
4. The last step is same as the step 3 of the transition coverage.

The result path is the test case satisfies the node coverage of activity diagram. The steps of 1, 2 are to reduce the number of transition and guarantee the connection of graph. Step 3 using original graph is to reduce the length of single path.

The automatic generating test cases for ATM example satisfy activity state coverage is shown as figure 5.

The corresponding prototype tool UMLTTC has been developed using language C#. Result shows, test cases generation for activity diagram base on Euler circuit provides an automatic and effective way to produce a set of test cases. This paper for the first time combines software testing with Extencies. Up to now, a series of particular extension methods have been developed. Extenics achievements have been successfully applied to many fields such as engineering science, information science, economics and management, and start producing economic and social benefits [1].
5. Conclusions and Future Study Directions

In software testing, we cannot test all paths of the system for limited time and limited cost. This paper considers combining software testing with Extencies and proposes an automatic approach to generating reduced test cases from UML activity diagrams partly using Extension Theory. We attempt to formalize the activity diagram using n-dimensional matter-elements. In order to maximize the faults detection by minimized test cases, we transform the activity diagram to the Euler circuit based on transformation theory. Finally, test cases are generated automatically by Euler circuit algorithm. Here, we decrease the number of test cases for the Euler circuit intrinsic character. We have developed a prototype tool to generate test cases using this method for UML activity diagram. Some examples have been executed in our tool. Result shows, this method provides an automatic and practical way to produce a set of reduced test cases.

Now, we only focus on the basic test coverage criteria like activity state coverage and transition coverage. Our future work is considering how to generate test cases satisfy more test coverage from UML diagrams. We hope this approach can do some help to the automatically test cases generation from UML diagrams and the development of the Extencies.

Acknowledgements

Thanks for some implementation work by bin Zhang and Jinsen Lin etc.

This research has been supported by grants from National Natural Science Foundation of China (#71271191, #70921061, #70871111), Bureau of Science and Technology of Ningbo.

References


