Role Oriented Test Case Generation for Agent Based System

N.Sivakumar, K.Vivekanandan, A.Mohan
Department of Computer Science and Engineering
Pondicherry Engineering College, Puducherry, INDIA
sivakumar11@pec.edu, k.vivekanandan@pec.edu, mohan4226@pec.edu

***Abstract*— Agent Oriented Software Engineering (AOSE) is a rapidly developing area of research. Current research and development primarily focuses on the analysis, design and implementation of agent based software whereas testing is less prioritised. Software testing is an important and indispensable part of software development process. Test case generation is the primary step of any testing process which is followed by test execution and test evaluation. Test case generation is not an easy task but upon automating the test case generation process serves many advantages such as time saving, effort saving and more importantly reduces number of errors and faults. This paper investigates about generating test cases for testing agent based software. We propose a novel approach, which takes advantage of agent's role as the basis for generating test cases. Role is an important mental attribute of an agent which is simply defined as the set of capabilities that an agent can perform. The main objective of this paper is to generate test cases from role diagram upon converting it to activity diagram.**

Keywords- Software Agent, Automatic Test Case Generation, Role Diagram, Agent Oriented Software Engineering

I. INTRODUCTION

In recent years, agent-based systems have received considerable attention in both academics and industry. Agent based systems are new paradigm for modelling and building complex distributed system. The agent-oriented paradigm can be considered a natural extension to the object-oriented (OO) paradigm. A Multi-Agent System (MAS) is a software system made up of multiple independent and encapsulated loci of control (i.e., the agents) interacting with each other in the context of a specific application viewpoint [1]. The importance of testing and debugging is universally accepted and recognized. Research on MAS testing is still at the early stage and very few work have been undertaken to provide tool support for testing activity [2]. Agent behaves differently at different instance for same test case based on the environment need and to accomplish the intended goal. Thus executing same test case several times may yield different execution path and as a result different output depending on the execution path. This makes the generation of test cases more complex and at the same time, if successfully the test cases are generated, it is more advantages too. The major benefits of test case generations are time saving, effort saving and more importantly it reduces the number of errors and faults.

There are lots and lots of speculations over role definition but in simple terms, a role can be defined as the capability enabler that exposes to the agent that plays it a set of actions [3]. Roles are created to do something and it has the responsibility of achieving specific system goals and sub-goals. They are similar to the notion of an actor in a play or an office within an organization. For example, roles at a college includes "Head of the Department", "Teacher", "Researcher", "Class Advisor", "Student Counsellor" and many more. One person can hold one role (a person can take Teacher role), and one person can hold more than one role (a person holding Teacher role can also be a Head of the Department). Roles define the person's responsibilities and functions in an organization. Similarly in an agent-based system, the roles are defined as the responsibilities and functions of agents. The main objective of this paper is to generate test cases from role diagram. Role diagram is the representation of agents involved in the system and its corresponding roles and responsibilities so as to achieve the system's goal.

Model-Based Testing (MBT) [4] is a type of testing strategy that depends on extracting test cases from different model and more importantly Unified Modelling Language (UML) models are the most highly ranged types of models. UML is the most dominant and attractive modelling language that is used for modelling the requirements and considered to be an important source for test case design. UML plays a major role in reducing cost and effort of testing and more importantly improves the quality of the software. Activity diagram is one of the important UML models to generate test case. Activity diagram represents the workflows of step by step activities that are utilized to describe the business and operational components of the system. The main advantage of using activity diagram is its simplicity and ease of understanding the flow of logic of the system.

The roles of the agents and its interaction with other agents are represented using role diagram. The paper aims at generating test cases from role diagram. To achieve the objective of generating test cases from role diagram, the first step is to convert the role diagram into activity diagram (i.e) every role corresponding to the

agent is mapped towards the activity of an agent. The activity diagram is used to create a directed graph which is called as Activity Dependency Graph (ADG). The ADG is examined using the Depth first Search (DFS) in order to extract all possible activity path and thereby the required test cases are derived.

II. RELATED WORK

A. Test Case Generation for MAS Using Formal Specification[5]

This work describes the formal approach for specifying, developing and testing multi-agent systems. This approach is based on the use Maude algebraic language which facilitates the description and representations of relations between elements in a transparent manner with regards to the internal behaviour of each agent class.

B. Ontology based Test Generation for MAS[6][7]

This work proposes a novel approach, which takes advantage of agent interaction ontologies that define content semantic of agent interactions to: (i) generate test inputs; (ii) guide the exploration of the input space during generation; and, (iii) verify messages exchanged among agents with respect to the defined interaction ontology.

C. Goal Oriented Test Case Generation for MAS[8]

A very natural way of testing the achievement of a goal is to check one's work or behaviour with respect to the goal. For a Multi agent system, in order to test a goal, we have to check what the system does or plans to do to fulfil the goal. The relationship between goal and plan plays a major role in finding out how goals can be fulfilled. Developers derive test suite from goal diagrams by starting from the relationships associated with each goal. Each relationship gives raise to a corresponding test suite, consisting of a set of test cases that are used to check goal fulfilment as well as unfulfillment.

D. Test Case Generation for UML Diagrams

There are plenty of works on generating test cases from UML diagrams such as State diagram [9], Sequence diagram [10], Activity diagram [11], Class diagram [12]. All these generation techniques are meant for testing object-oriented system. Test cases from object diagram are derived by analysing the dynamic behaviour of the objects due to internal and external stimuli. Test cases are generated from UML activity diagram by converting activity diagram into directed graph thereby generating test cases for the independent path identified from the directed graph.

III. PROPOSED WORK

More than one agent interacts and co-ordinates among one another to form a Multi-Agent System (MAS). The main objective of the proposed work is to generate test cases to test an agent based system using agent's mental attribute namely *Role*. Every agent has its own intended goals and to achieve its goal, it has to take several roles. Role might be defined as set of activities or responsibilities. Thus to ensure whether the roles are rightly handled by their corresponding agent, it is more important to verify the responsibilities of each role [13]. A role model representation is given in figure 1.

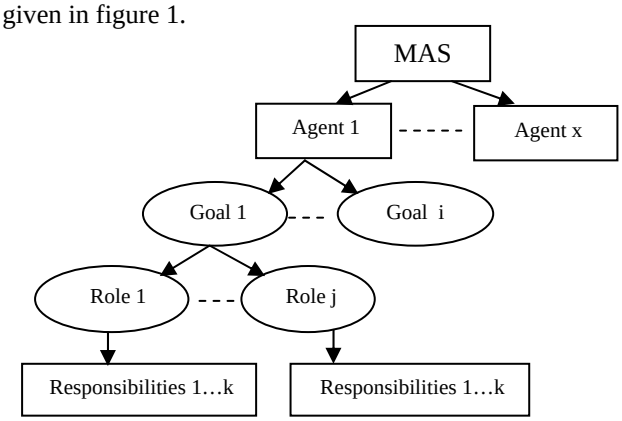

Fig 1. Role Model

To achieve role based test case generation, the roles to be performed by various agents involved in the Multi-Agent system are analysed and represented as role diagram. Role diagram depicts the roles carried out by appropriate agent so as to achieve the intended goal. Every step involved in achieving the role can be mapped to an activity and the representation is given accordingly. Thus from the role diagram of the MAS, an activity diagram is figured out. Test cases can be generated from the activity diagram, which represents the expected

structure and behaviour of MAS under testing. The following are the steps involved in role based test case generation,

1. Representing the role diagram with necessary test information such as agents involved, roles of agents, responsibilities of an agent, Agent goal.
2. Converting the role diagram into UML based activity diagram with necessary test information.
3. Representing Node Description Table (NDT) for Activity Graph.
4. Finally the ADG with the ADT are used to generate Activity Path
5. Deriving test cases for Activity Path
6. Validating the generated test cases

Our proposed role based test case generation technique can be explored with the help of a case study namely agent based online shopping system. Online shopping system is composed of several agents such as Buyer agent, Seller agent, Customer agent, Admin agent. For our illustration, we have taken buying scenario use case of agent based online shopping system.

The role model given in fig.1 clearly represents the hierarchical nature of the MAS implementation. The approach handled here is bottom-up approach i.e roles handled by the agents are tested to ensure that these roles are focused towards the goal it was intended. Thus test cases has be written and executed to test the roles. Test case generation is the primary step of any testing process which is followed by test execution and test evaluation. Test case generation is not an easy task but upon automating the test case generation process serves many advantages such as time saving, effort saving and more importantly reduces number of errors and faults. The first step of our proposed role oriented test case generation is to represent the role diagram of the multi agent system (i.e) Agent based online shopping system

With respect to customer's perspective, buying a product successfully is a goal. To achieve this goal, the following roles have to be performed by the agents namely login, browse product, find the product, check availability, add to cart, payment and shipping. The following table gives the possible roles that can be hold by the agents involved the MAS to attain the goal i.e buying the product.

TABLE I. ROLE DESCRIPTION TABLE

Test Suite	Goal	Agent	Role
TS1	Buying a Product	Customer Agent	User Registration Login
TS2		Buyer Agent	Browse Product Find Product Find Seller Check Availability Price Negotiate Adding to cart
TS3		Seller Agent	Add Product Payment Shipping

From Table.1, it is understood that, to achieve the system goal i.e buying a product, the agents involved in the system such as customer agent, buyer agent and seller agent have to take appropriate role. In this scenario, Customer agent holds user registration role and login role, whereas Buyer agent takes Browse product role, Find product role, Find seller role, Check availability role, Price negotiation role & Adding to cart role and Seller agent takes Add product role, Payment role and Shipping role. Unless or other than every agents play their roles the system goal cannot be achieved. In this regard, agent's role plays a major part in the system. Thus testing an agent system based on agent's role attribute will be more appropriate and effective too.

A. Role Diagram

To test the agent based system based on role, it is very important to know what agent hold what roles. Roles are nothing but how agent reacts to the input from the environment [14]. To represent the roles that have been played by different agent at different instances can be represented in a role diagram. Role diagram represents goal, sub-goal, and appropriate roles. Figure.2 depicts the role diagram of agent based online shopping system in which only the goals and roles of customer agent alone is represented. From the diagram, it is understood that the goals of customer agent is to facilitate home page access for existing user as well as new user. New user has to create his account by providing his personal details and opt for a username and password. Whereas existing

user is the one who already possesses his own username and password. The existing user has been given the option of deleting his account, edit some changes in this account and update his account. Login role is the one in which existing user directly logins and browse the product in the market. Existing user can also retrieve password if the password was forgotten during login by providing secret question and answer. Also existing user has been facilitated to change his password after login.

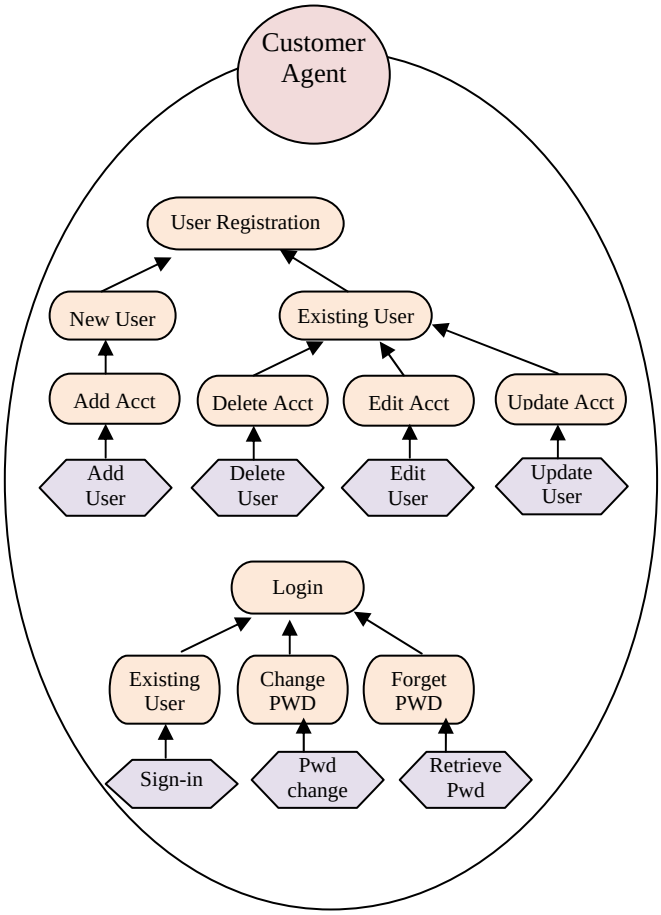

Figure 2. Role Diagram

B. Converting Role diagram into Activity Diagram

Role diagram represents the roles handled by various agents involved in the Multi-Agent system. Role can be defined as the set of activities handled by an agent to achieve the intended goal. From the role diagram depicted in figure 2, it is understood that to accomplish new user registration role, the activities like add account, delete account, edit account and update account have to processed. Add account activity in turn involves many steps such as providing username, password, personal details like name, address, e-mail etc., Add account activity also ensures the validity of the data provided by the users. Validation process is to check whether the username starts with a alphabets, to check whether the username is available already whether password has minimum 8 characters etc., The following figure.3 depicts the new user registration role in which adding new account activity is deliberated.

C. Converting Activity Diagram into Activity Graph

Graph G (V, E) is defined as the set of vertices (V) connected by edges (E). An activity graph is a directed graph, where each node represents a constructs (initial node, decision node, final node etc.,) and each edge represents the flow in the activity diagram. Every constructs of activity diagram is mapped to a node and the flow that connects the constructs forms the edges. Figure.4 depicts an activity graph mapped from activity diagram (figure.3) of new user registration. For our reference we label the nodes of the activity graph from 1 (start node) to 21(stop node).

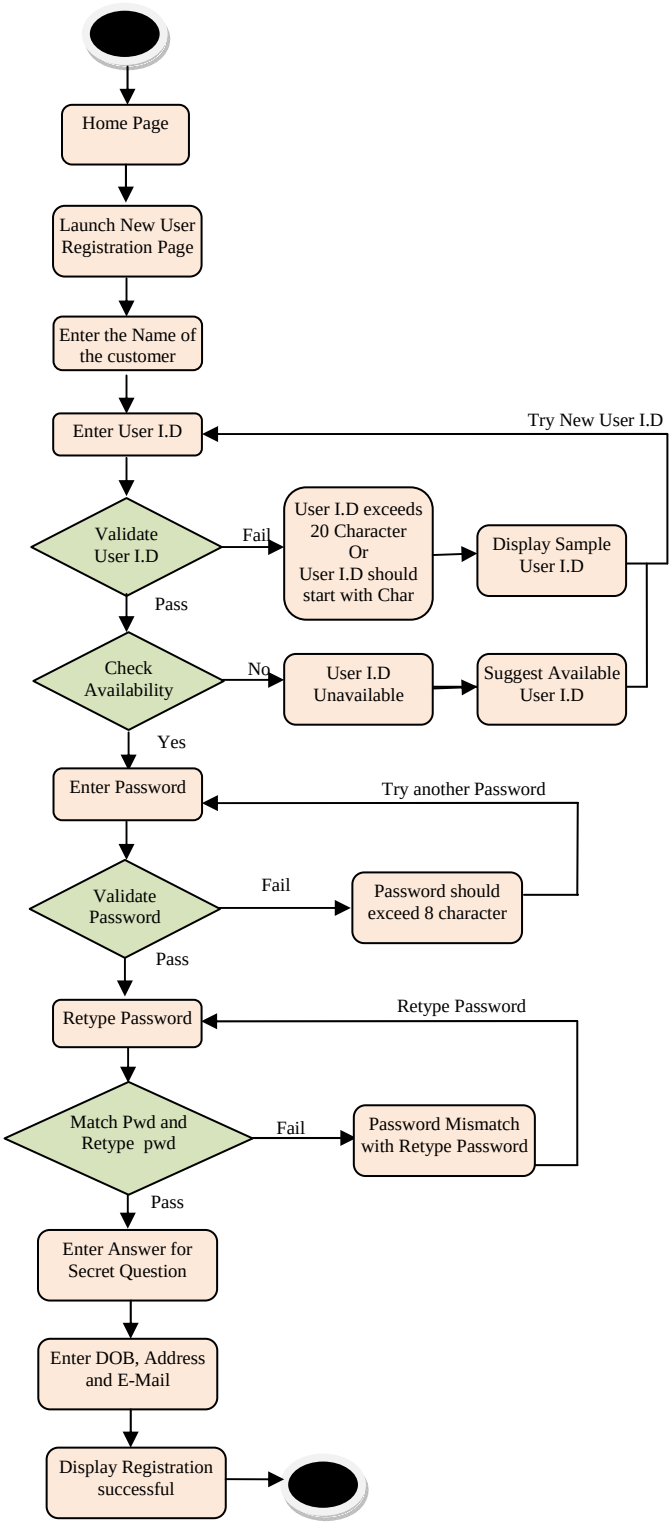

Figure.3 Activity Diagram of New User Registration

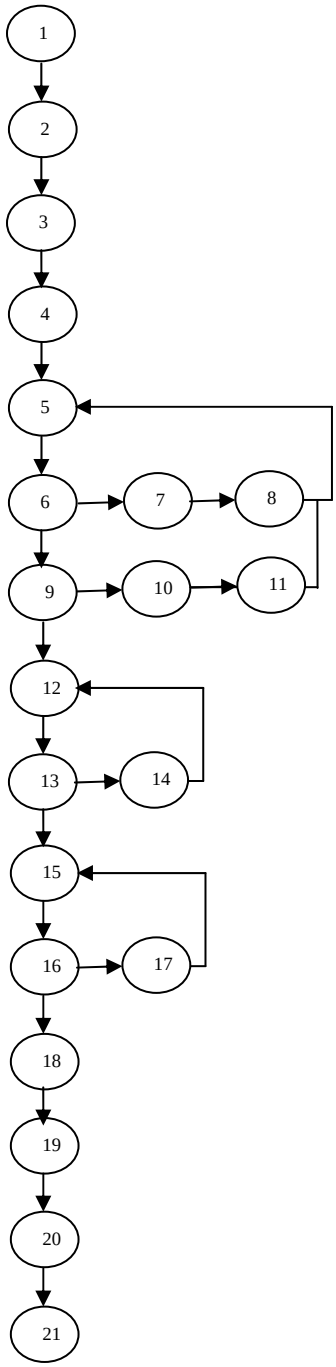

Figure.4 Activity Graph of New User Registration

D. Node Description Table (NDT) for Activity Graph

For easy references, we store the detailed information of each node in the activity graph in a table named as Node Description Table (NDT). The following Table.1 represents the node index and its corresponding activity name.

TABLE II. NODE DESCRIPTION TABLE

Node Index	Associated Activity Name / Branch Condition
1	Start
2	Home Page
3	Display New User Registration Page
4	Enter The Name of the User
5	Enter User I.D
6	Validate User I.D
7	User I.D Validation Failed
8	Display Sample User I.D Format
9	Check User I.D Availability
10	User I.D Unavailable
11	Suggest Available User I.D
12	Enter Password
13	Validate Password
14	Password Exceeds Minimum Character
15	Retype Password
16	Matching Password and Retyped Password
17	Retyped Password Mismatched with Password
18	Enter Secret Question and Answer
19	Enter DOB, Address and E-Mail
20	Registration successful
21	Stop

E. Generating Activity Path from Activity Graph

Path coverage criterion is the basis for generating test cases from activity diagram. The first step in generating test cases is to obtain all possible activity paths from the node start to stop in the activity graph. To identify the paths from the activity graph, Breadth First Search (BFS) and Depth first Search (DFS) traversal techniques are incorporated in the algorithm [15]. An activity graph is given as the input to the algorithm and it results with the activity paths.

Input : An activity graph

Output: Set of activity paths

1) Initialize: LoopFlag=0; Visits of every node=0; Stack S and queue Q is used for keeping the track of visited nodes.
2) Begin
3) Traverse the activity graph using depth-first search (DFS). For each node visited during the traversal, count its no of visits and push the node into the stack S;
4) if Visits of the current node = 2 then
5) Set LoopFlag=1;
6) end
7) if LoopFlag=1 and visits of current node = 3 then
8) Backtrack to the node (of type 'D') which has at least a child node with its visits less than two, and then repeatedly pop from S until top of S is the node where recent backtrack has stopped;
9) end
10) if Type of currently visited node NF ='F' then

11) Traverse the subtree rooted at node NF using breadth-first search (BFS) and enqueue the nodes of type 'A' into Q until node of type 'J'/'E' is visited for out-order-degree (NF) number of times; At end, enqueue the node of type 'J' found during recent breadth-first-search(BFS) into Q;
12) while Q is Not Empty do
13) Dequeue q from Q;
14) if (type(q)!='J') then
15) Push q into S ;
16) Explore all descendent dq of node 'q' with taking 'q' as the root upto maximum depth=2 using depth-first-search (DFS) traversal and push dq into S if type of dq is other than 'A' and 'E' and 'J';
17) end
18) end
19) end
20) if Type of currently visited node='E' then
21) Copy current content of S into an array where top of S represents last node of activity path, sequence of elements in the array is an activity path; Backtrack to node of type='D' that has at least a child node whose visit count=0, and then repeatedly pop from S until top of S is the node where recent backtrack has stopped; visit its child node whose visit count = 0; If S is found to empty, stop else go to step 3 ;
22) end
23) End

Activity graph (figure.4) is given as input to the above mentioned algorithm and the following independent path from the activity graph are obtained as the output

1) 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8
2) 1 – 2 – 3 – 4 – 5 – 6 – 9 – 10 – 11
3) 1 – 2 – 3 – 4 – 5 – 6 – 9 – 12 – 13 – 14
4) 1 – 2 – 3 – 4 – 5 – 6 – 9 – 12 – 13 – 15 – 16 – 17
5) 1 – 2 – 3 – 4 – 5 – 6 – 9 – 12 – 13 – 15 – 16 – 18 – 19 – 20 – 21

The identified paths are described based on the Node Description Table (NDT). Identified path are described as follows,

1) 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8

Start – Home Page – Launching New User Registration Page – Enter the name of the Customer – Enter User I.D – Validate the User I.D – If (User I.D > 20 Character and User I.D start with Numeric) then user I.D fails – Display Sample User I.D – Try Again

2) 1 – 2 – 3 – 4 – 5 – 6 – 9 – 10 – 11

Start – Home Page – Launching New User Registration Page – Enter the name of the Customer – Enter User I.D – Validate the User I.D - If (User I.D < 20 Character and User I.D start with Character) then Check User I.D Availability – If User I.D already Exists – Suggest available User I.D – Try Again

3) 1 – 2 – 3 – 4 – 5 – 6 – 9 – 12 – 13 – 14

Start – Home Page – Launching New User Registration Page – Enter the name of the Customer – Enter User I.D – Validate the User I.D - If (User I.D < 20 Character and User I.D start with Character) then Check User I.D Availability – If User I.D already Available – Type Password – Validate Password – If (Password < 8 character) – Try Another Password

4) 1 – 2 – 3 – 4 – 5 – 6 – 9 – 12 – 13 – 15 – 16 – 17

Start – Home Page – Launching New User Registration Page – Enter the name of the Customer – Enter User I.D – Validate the User I.D - If (User I.D < 20 Character and User I.D start with Character) then Check User I.D Availability – If User I.D already Available – Type Password – Validate Password – If (Password > 8 character) then Retype Password – Match Password and Retyped Password – If Password and Retyped Password Mismatches then Try Again

5) 1 – 2 – 3 – 4 – 5 – 6 – 9 – 12 – 13 – 15 – 16 – 18 – 19 – 20 -21

Start – Home Page – Launching New User Registration Page – Enter the name of the Customer – Enter User I.D – Validate the User I.D - If (User I.D < 20 Character and User I.D start with Character) then Check User I.D Availability – If User I.D already Available – Type Password – Validate Password – If (Password > 8 character) then Retype Password – Match Password and Retyped Password – If Password and Retyped Password matches Enter Secret question and answer – Enter DOB, Address and E-Mail – Display Registration Successful

F. Cyclomatic Complexity

The number of activity path identified by the algorithm can be validated by computing cyclomatic complexity. Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program. When used in the context of the basis path testing method, the value computed for cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.

Cyclomatic complexity has a foundation in graph theory and provides us with extremely useful software metric. Complexity is computed in one of three ways:

1. The number of regions of the flow graph corresponds to the cyclomatic complexity.
2. Cyclomatic complexity, $V(G)$, for a flow graph, G, is defined as

$$V(G) = E - N + 2$$

where E is the number of flow graph edges, N is the number of flow graph nodes.

3. Cyclomatic complexity, $V(G)$, for a flow graph, G, is also defined as

$$V(G) = P + 1$$

where P is the number of predicate nodes contained in the flow graph G.

Referring to the flow graph in Figure.3, the cyclomatic complexity can be computed using each of the above mentioned computation:

1. The flow graph has 5 regions.
2. $V(G)$ = 26 edges –21 nodes + 2 = 5.
3. $V(G)$ = 4 predicate nodes + 1 = 5.

G. Deriving Test Cases

After identifying the activity path, data should be chosen so that conditions at the predicate nodes are appropriately set as each path is tested. Test cases that satisfy the basis set just described are

Path 1 test case:

Input

Value (user i.d) = invalid input, if user i.d > 20 char && user i.d starts with numerals

Expected results

Display sample user i.d and Try Again

Path 2 test case:

Input

Value (user i.d) = valid input and value (user i.d) = unavailable

Expected results

Display few available user i.d and Try Again

Path 3 test case:

Input

Value (user i.d) = valid input && value (user i.d) = available && value (password) = invalid (password length < 8)

Expected results

Try again with password whose length > 8 character.

Path 4 test case:

Input

Value (user i.d) = valid input && value (user i.d) = available && value (password) = valid && Retype password = Mismatch with original password

Expected results

Retype password and Original password should be same. Try again

Path 5 test case:

Input

Value (user i.d) = valid input && value (user i.d) = unavailable && value (password) = valid && Retype password = matches with original password

Expected results

Successful registration of user

TABLE III. GENERATED TEST CASES

Test Case No	Input	Activity Sequence
1	User I.D = Invalid.	Start, Launching New User Registration Page – Enter the name of the Customer – Enter User I.D – Display "Invalid User I.D"
2	User I.D = valid. User I.D Availability = No	Start, Launching New User Registration Page – Enter the name of the Customer – Enter User I.D – Check for Availability – Display " User I.D already Exist"
3	User I.D = valid. User I.D Availability = Yes Password = invalid	Start, Launching New User Registration Page – Enter the name of the Customer – Enter User I.D – Check for Availability – Enter Password – Display " Invalid Password"
4	User I.D = valid. User I.D Availability = Yes Password = valid Retype Password = Mismatch	Start, Launching New User Registration Page – Enter the name of the Customer – Enter User I.D – Check for Availability – Enter Password – Retype Password – Display " Retype Password should be the same; Please Retype Correctly"
5	User I.D = valid. User I.D Availability = Yes Password = valid Retype Password = match	Start, Launching New User Registration Page – Enter the name of the Customer – Enter User I.D – Check for Availability – Enter Password – Retype Password – Enter Secret question and answer – Enter DOB, Address, E-mail – Display " Registration completed"

Our proposal of role oriented test case generation targets to detects the faults such as fault in decision, fault in loop and synchronization fault.

1) Fault in decision

This fault may happen in decision making scenario. For example, a fault in the decision which validates the username may display invalid username though entered valid username.

2) Fault in loop

This fault may happen during loop entry or loop exit or increment/decrement operation. Say for example, during new user registration, when password and retyped password mismatches then loop should be iterated that facilitates *Try Again* option

3) Synchronization fault

This fault might happen when some activity begins execution before completion of preceded activities. For example, displaying *Registration Successful* should not happen before keying user detail activities.

IV. CONCLUSION

Agent technology has drawn much attention as the preferred architectural framework for the design of many distributed software systems. Agent-based systems are often featured with intelligence, autonomy, and reasoning. Such attributes are quickly becoming alluring to both legacy and new systems. Agents are building blocks in these software systems, while combinations of attributes are composed to form the software entities. Testing an agent is a complex task as agents may behave differently for same input at different instances. In such case deriving test case in a time consuming process for testing agent based system. In this paper, role oriented test case generation technique for testing an agent based system is proposed. Role is an important attribute of an agent and it is defined as capability enabler or set of activities carried out by an agent to achieve a goal. The proposed work derives test cases from the role diagram which is nothing but the descriptive representation of all the roles carried out by respective agents to achieve their corresponding goals. The role diagram is converted into UML based activity diagram from which the test cases are derived using activity graph and node description table. Cyclomatic complexity is computed to validate the paths identified from the activity graph. Finally test cases are derived in such a way that the predicate nodes are appropriately set as each path is tested. The proposed role based test case generation technique detects faults such as fault in decision, fault in loop and synchronization fault as well as ensures statement coverage, loop coverage and code coverage.

REFERENCES

[1] Nwana.G, "Software Agents: An Overview", The Knowlwdge Engineering Review, 11(3).

[2] Mailyn Moreno, Juan Pavon, Alejandro Rosete. Testing in Agent Oriented Methodologies. IWANN 2009, Part II, LNCS 5518, pp. 138–145, 2009.© Springer-Verlag Berlin Heidelberg 2009

[3] Haiping Xu, Xiaoqin Zhang, Rinkesh J. Patel. Developing Role-Based Open Multi-Agent Softwrae Systems. International Journal of Computational Theory and Practice Vol.2, No. 1, June 2007.

[4] Zhiyong Zhang, John Thangarajah, Lin Padgham, " Model Based Testing for Agent Systems (Extended Abstract)", Proc of 8th Int. Conf on Autonomous Agents and Multiagent Systems (AAMAS 2009), Hungary, pp.1333-1334.

[5] Yacine Kissoum and ZAidi Sahnoun, "Test Case Generation for Multi-Agent Systems Using Formal Specification" AICCSA 2007: 76-83.

[6] Manjeet kumar. Roles and Ontology for Agent Systems. Global Journal of Computer Science and Technology Volume 11 Issue 23 Version 1.0 December 2011

[7] Cu. D.Nguyen, Anna Perini and Paolo Tonella, " Ontology-based Test Generation for Multi Agent Systems (Short Paper), Proc. Of 7th Int. Conf. on Autonomous Agents and Multiagent Systems, May 2008, Portugal, pp.1315-1318

[8] Duy Cu Nguyen, Anna Perini, Paolo Tonella. A goal-oriented software testing methodology. AOSE'07 Proceedings of the 8th international conference on Agent software engineering Springer –Verlag Berlin, Heidelberg-2008.

[9] S. Kansomkeat, W. Rivepiboon. "Automated-Generating Test Case Using UML State chart Diagrams", Proceedings of the 2003 annual research conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through Technology (SAICSIT), Republic of South Africa, 2003.

[10] M. Sarma, D. Kundu, R. Mall. "Automatic Test Case Generation from UML Sequence Diagrams", Proceedings of the 15th International Conference on Advanced Computing and Communications, IEEE Computer Society Washington, DC, USA, 2007.

[11] H. Kim, S. Kang, J. Baik, I. Ko. "Test Cases Generation from UML Activity Diagrams", Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD), Qingdao, China, 2007.

[12] M. Priestley. Practical Object-Oriented Design with UML. McGraw-Hill Education, Berkshire, UK, 2nd edition, 2003.

[13] N.Sivakumar, K.Vivekanandan, "Testing an Agent Based E-Novel System-Role Based Approach" Intelligent Informatics, AISC 182, pp. 165–173. Springer-Verlag Berlin Heidelberg 2012

[14] Haiping Xu, Xiaoqin Zhang, Rinkesh J. Patel. Developing Role-Based Open Multi-Agent Software Systems. International Journal of Computational Theory and Practice Vol.2, No. 1, June 2007.

[15] Debasish Kunda and Debasis Samanta, " A Novel Approach to Generate Test Cases from UML Activity Diagram", Journal of Object Technology, vol.8, no.3, May-June 2009.