# Software Test Data Generation Based On Multi-Agent

Siwen Yu , Jun Ai
Department Of System Engineering of Engineering Technology, Beihang University, Beijing 100191, China
applifish126@gmail.com, aijun@buaa.edu.cn

**Abstract.** Software test data generation is an important part of software testing. This paper presents a multi-agent cooperation framework for software test data generation. The framework is constituted by Graph Miner, Method Selector and test data generation method agent group. Graph Miner extracts software information sequences from UML graphs, and sends them to Method Selector. Then Method Selector selects out relevant method agents to generate test data. This framework can solve the problems of test data generation methods extension and low level of intelligence exist in traditional methods. Based on the framework proposed in this paper, a software prototype is developed. It is proved that this framework is feasible.

**Keywords:** Agent; software test data; UML

## 1 Introduction

UML is a semi-formal modeling language, which is well defined and easy to express. It is an effective resource to generate software test data. The UML based test data generation method is a type of black box testing method. The black box testing method includes equivalence class classification, boundary value analysis, causal mapping, statistical test method, pairs test method and various test methods based on artificial intelligence. But the traditional UML based software data generation methods are poor to extend and can not impose a variety of testing methods, which can not be effective in the discovery of software defects.

Agent refers to the entity present in a particular environment to carry out its own action to meet the goal, with the characteristics of scalability, flexibility and high degree of autonomy. By setting the relevant rules, Agent can impose a wide range of software test data generation methods in accordance with the actual needs. By adding the rules, new software test data generation method can be added. This overcomes the shortcomings of poor scalability and method oneness. This paper uses the cooperation of the multi-Agent to complete the generation of software test data.

## 2 The Principles of the Framework

The framework proposed by this paper can be described by the following formula: $S=\{U, C, D, V, f\}$[1]. $U$ represents UML models. Fig1 shows the details of the framework. $R=C \cup D$ represents the property set of the framework, in which $C$ is the conditional properties and $D$ is the result condition. $V$ is the value set of property, $V=\cup v_r$, $r \in R$. f is information function, $f$: $U \times R \rightarrow V$. $f$ defines how to select test data generation method in accordance to the specified UML diagrams.

Based on the UML model $U_p$, the framework can choose the test data generation methods agent set $D$ according to conditional property $C$. The agents which is selected are then to execute method to generate test data..

According to the process of test data generation, the principles above can be summarized into the following three steps:

1. Extract software test information from UML diagrams;
2. Select agent set $D_i$ from rule set $C_i$;
3. Execute the selected agents to get software test data.

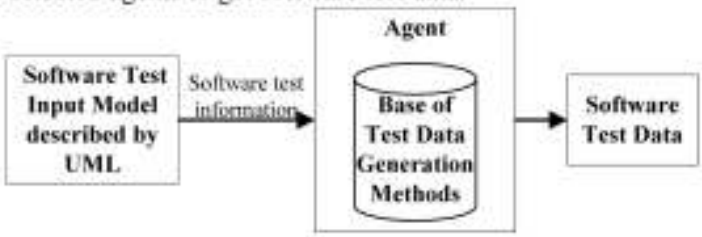


**Fig1** The framework of agent based software test data generation

This paper introduced in two agents and one agent group. They are Graph Miner, Method Selector and test data generation method agent group. There are many kinds of method agent in test data generation method agent group. Graph Miner extracts test information from UML diagrams to provide information to the generation of test data. Through the knowledge base in Method Selector, the agent $k$ is selected from the test data generation method agent group according to $C$. Then agent $k$ is executed to generate test data. Fig 2 shows framework of the design.

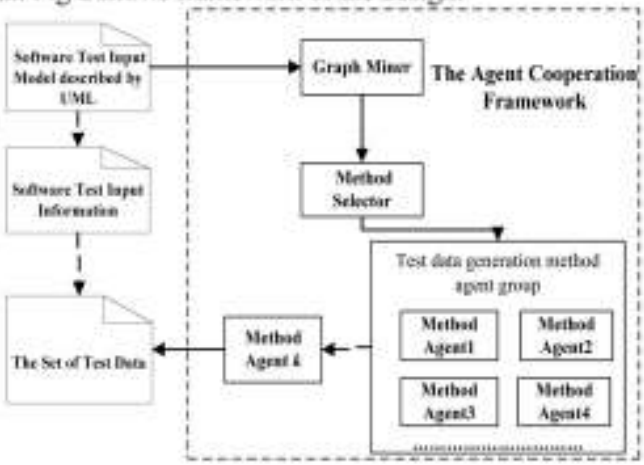

**Fig 2** Agent cooperation framework

## 3 Test Input Information Extraction

It is necessary to clarify the concept of software test input information. For this, the paper defines software testing input package and software testing information sequence.

**Definition 1** (Software testing input package): Let *InputPackage* represents software testing input package, which is a triple vector. *InputPackage*={*Var*, *VarType*, *VarRange*}, the definition of elements in the vector is as follows:

1. *Var*: variable name in the test input;
2. *VarTye*: variable type in test input;
3. *VarRange*: variable scope in test input.

**Definition 2** (Software testing information sequence): Software testing information sequence is a quaternion group *STS*={*Start*, *Final*, *Restriction*, *InputPackage*}.The meaning of the elements in the vector is as follows:

1. *Start*: the start point of the UML diagram;
2. *Final*: the final point of the UML diagram;
3. *Restriction*: the conditions of the migrations;
4. *InputPackage*: oftware testing input package.

The content and format of software testing information sequence has nothing to do with the specific UML diagrams. This provides the information source to generate test data.

### 3.1 The coverage criteria of UML diagrams

In order to ensure the adequacy of software testing, we must consider the validity of test data and coverage criteria. Based on the concept of the software testing information sequence, the constraints, nodes and variable information should be extracted from the diagrams. So, this paper defines criteria to cover these elements. The criteria are as follows:

1. Constraint coverage criterion: cover all the constraints in the UML diagrams;
2. Node coverage criterion: cover all the nodes in the UML diagrams;
3. Variable information coverage criterion: cover all the information of variable in the UML diagrams, such as variable name, variable scope and variable type.

### 3.2Extraction of software test input information

Graph Miner traversals UML diagram *Up* to generate software testing information sequence in accordance with the test coverage criteria. The specific process is as follows:

1. Read the type mark of the UML diagram $T_p$;

2. Select UML diagram traversal methods based on mark $T_P$.
3. Implement traversal method rules, obtain the software testing information sequence.

Through the process, the software testing information sequence *STSp* can be generated from model *Up*. The following is the algorithm to generate software testing information sequence from UML activity diagram.

```
typedef struct migration{
   int adjvex;//the position of the node
   string Restriction;//restrict condition
   struct migration *next;//point of the next edge
       }Elink;
typedef struct node{
    string var;//name of variable
    string varitype;//type of variable
    int varrange[M];//range of variable
    Elink *link;}VNode;
```

Algorithm: Generate test information of activity diagram

Input: UML activity diagram T

Output: Test information of activity diagram

Algorithm design:

```
travel_Activity_Diagram(VNode G[], int vnode)
  {  int w;
     Visit(vnode);  //visit the information in vnode,
                    extract test information
      w=FIRSTNODE(G,vonde);
     //look for the first adjacent point of V,
       return -1 if no adjacents exist
     while (w!=-1)
     {if(vistited[w]= =0)
     travel_Activity_Diagram(G,w);
     w=NEXTNODE(G, vonde);
     // look for the next adjacent point of V,
       return -1 if no adjacents exist
     }}
```

## 4 Software test data generation

After the software testing information sequence is generated, Method Selector chooses the relevant software test generation method Agent to generate software test data. Method Selector completes the task by the knowledge base of software test data generation method internal. In fact, Method Selector is the executor of the mapping $U \times R$ to $V$.

The value table $Y$ of $R$ is in the Method Selector. The generation of test data is based on the software testing information sequence. Therefore, the elements of the sequence should be the conditional elements for the test data generation. In this paper, table $Y$ is realized by the table of software test information sign-Agent name-Agent ID (Table TST-AN-AID for short). In table 1, $B_i$, $i \in N$ represent the set of the name of

the Agents be selected. $Bi=\{Lij|i,j\in N$, $Lij$ is the name of the method$\}$. $Agi$, $i\in N$ represent the set of IDs of the Agents. $Agi=\{Gij\in G \mid G$ is the set of ID of the Agents$\}$.

**Table 1** TST-AN-AID table

| *C* | *D* | |
|---|---|---|
| Test information sign | Agent name | Agent ID |
| *Var:T1* | *B1* | *Ag1* |
| *VarType:T2* | *B2* | *Ag2* |
| *VarRange:T3* | *B3* | *Ag3* |
| *Start:T4* | *B4* | *Ag4* |
| *Final:T5* | *B5* | *Ag5* |
| *Restriction:T6* | *B6* | *Ag6* |

Based on the table of TST-AN-AID, test data generation method agent *Bi* is selected out to generate test data. The steps are as follows:

1. Graph Miner extracts software testing information sequence from UML diagrams;
2. Trough TST-AN-AA table software test data generation Agent *k* is selected out;
3. Agent *k* generates software test data based on the software testing information sequence.

In order to complete the cooperation between the agents, this paper uses Indirect / Active termination model [2]. The principle of the model is showed in Fig 3.

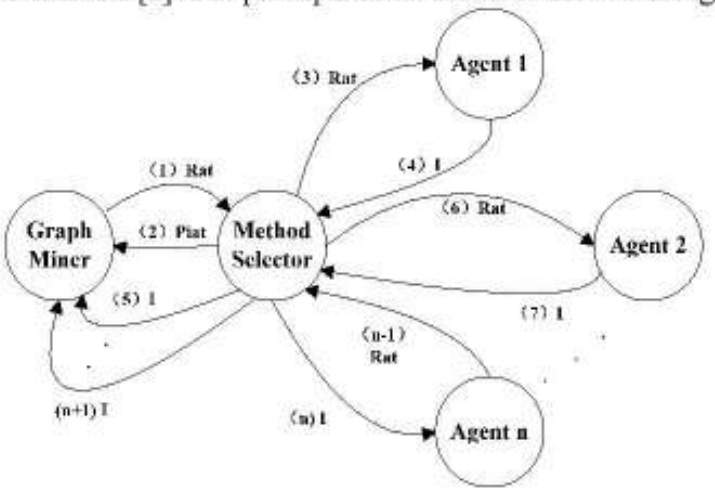


**Fig 3** indirect / active termination model

## 5 Case study

In this paper, an aircraft plug-weapon management system is used as an example to illustrate the principle. There are six use case diagrams, three activity

diagrams and one sequence diagrams. Figure 4 is the system's system-level use case diagram. Due to the restrictions on article length, can not all of the diagrams be listed.

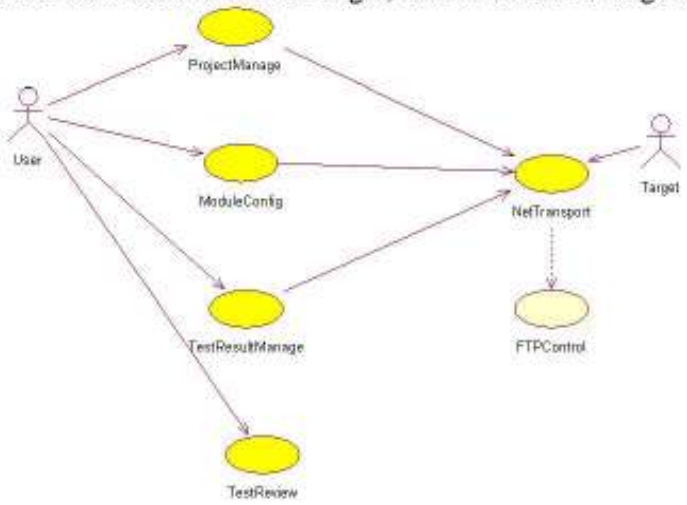


**Fig 4** system use case diagram

Graph Miner traverses through the use case diagram, to generate the software testing requirement tree C, Figure 5. The node types of Tree C are listed in Table 2.

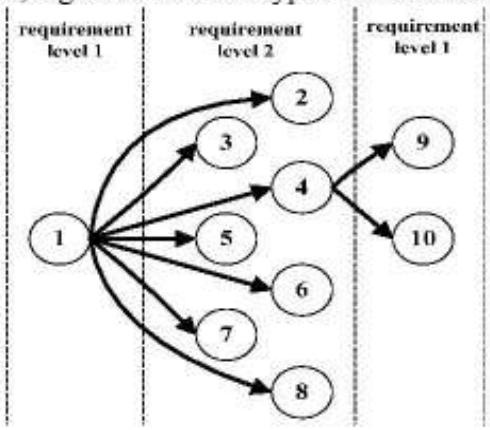


**Fig 5 Test requirement tree**

**Table 2** The diagram type and mark of test requirement tree

| Diagram type | Node | Diagram type mark |
|---|---|---|
| Use case diagram | 1、4、5、6、7、8 | $P_1$ |
| Activity diagram | 2、9 | $P_2$ |
| Sequence diagram | 3、10 | $P_3$ |

After the requirement tree is generated, the corresponding rules are found to generate software test information sequences through Table 3.

**Table 3** The generation of test information sequences

| Diagram type mark | Node | Rules | Software test information sequence |
|---|---|---|---|
| *P*1 | 1、4、5、6、7、8 | *Ap*1 | *STS* p1、*STS* p4、…、*STS* p8 |
| *P*2 | 2、9 | *Ap*2 | *STS* p2 |
| *P*3 | 3、10 | *Ap*3 | *STS* p3, *STS* p10 |

Based on the software test information sequence, test method agents are selected to generate test data through table 4.

**Table 4** The generation of software test data

| STS | Element | Agent name | Agent ID | Software test data set |
|---|---|---|---|---|
| *STS*1, *STS*2, .., *STS*10 | *Var* | *B*1 | *Ag*1 | *D*11~ *D*110 |
| | *VarType* | *B*2 | *Ag*2 | *D*21~ *D*210 |
| | *VarRange* | *B*3 | *Ag*3 | *D*31~ *D*310 |
| | *Start* | *B*4 | *Ag*4 | *D*41~ *D*410 |
| | *Final* | *B*5 | *Ag*5 | *D*51~ *D*510 |
| | *Rrestrictio n* | *B*6 | *Ag*6 | *D*61~ *D*610 |

The framework proposed by this paper can solve the problem of test data generation method extension. New method Agent can be added by add new elements in TST-AN-AA table. At the same time, the extension of test data generation method is realized. The principle is showed in Fig 6.

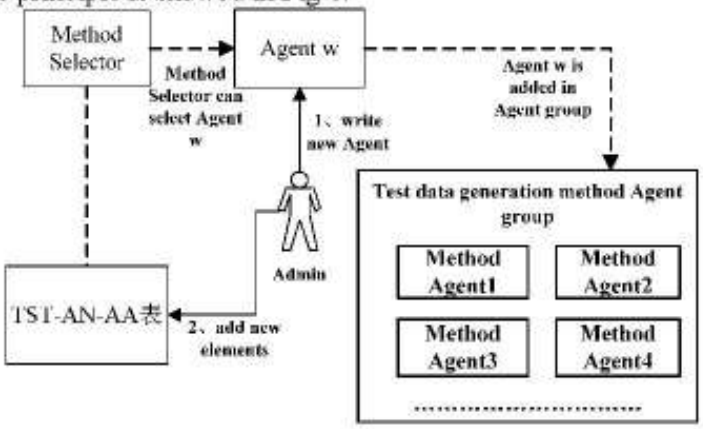


**Fig 6** principle of software test data generation method extension

## 6 Conclusion

Through using multi-Agent cooperation method in software test data generation, this paper proposes a framework of software test data generation. Based on the test

information extraction and test data generation, the principle and process of the Agent-based framework is elaborated. According to the framework proposed in this paper, a prototype is developed to verify the feasibility of the framework. In addition, the framework is easy to extend test method, can generate test data based on multiple UML diagrams and high level of intelligence.

## References

1. Wang Linzhang; Yuan Jiesong; Yu Xiaofenget et al, Generating test cases from UML activity diagram based on Gray-box method[J], Software Engineering Conference, 2004. 11th Asia-Pacific on 30 Nov.-3 Dec. 2004, Page:284 – 290.
2. Li, Bao-Lin; Li, Zhi-shu; Qing, Li et al, Test case automate generation from UML sequence diagram and OCL expression, Computational Intelligence and Security, 2007 International Conference on 15-19 Dec. 2007 Page:1048 – 1052.
3. Chevalley, Thevenod-Fosse, Automated generation of statistical test cases from UML state diagrams[J], Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International 8-12 Oct. 2001 Page:205 – 214.
4. Xiaoying Bai; Guilan Dai; Dezheng Xu et al, A multi-agent based framework for collaborative testing on Web services[J], Software Technologies for Future Embedded and Ubiquitous Systems, 2006 and the 2006 Second International Workshop on Collaborative Computing, Integration, and Assurance. SEUS 2006/WCCIA 2006. Page:61-67.
5. Zen Huanglin, Intelligent Computing[M], Chongqing University Press204: 2-4.
6. Mao Xinjun, Agent Oriented Software Development[M],Beijing: Tsinghua University Press2005:140.