A Program Testing Method for PDES Application

Tianlin Li[1, a], Yiping Yao[1], Feng Zhu[1], Huilong Chen[1]

[1]College of Information System and Management, National University of Defense Technology, Changsha, 410073, China

[a]email: ltl@mail.ustc.edu.cn

Keywords: PDES; automatic testing; test code; test case

Abstract. As PDES (Parallel and Discrete Event Simulation) applications become larger and more complex, program test for these applications becomes quite difficult. Current test approaches mainly rely on adding test code into program manually to record model data and to execute the given test cases, which are quite time-consuming. To solve this problem, a test method PDES-TCG based on RUM (PDES Test Code Generation) is proposed, which can be used to generate executable test code and insert them into application programs automatically. The test code can record run-time data of simulation models and test simulation entities with given test cases. The method is validated using a simple PDES application.

Introduction

PDES[1] applications usually contain multiple simulation entities, entity contains multiple simulation models, besides there are interactions between the entities. These conditions make applications very complex, so program errors are inevitable, program testing must be carried out to discover these errors. How to find these errors are the most concerned issues for testers. Adding test code into programs is a common method. Ideally, the proportion of test code and production code is 1: 1[2]. Adding test code manually will increase test workload, automatic generation of test code can be an effective solution to this problem. Test code generation technology has made some achievements, however the current study pay little attention on PDES application.

Reusable simulation model development Specification RUM[3] can reduce the difficulty of model integration and improve the development efficiency of PDES application. In addition, RUM provides a basis for automatic code generation of simulation models.

This paper proposes a test method for PDES application based on RUM called PDES-TCG, which can be used to generate executable test code automatically and insert them into the simulation program. PDES-TCG can be used as a general PDES application test method. In this paper, the method will be described by PDES application developed in C ++ language. Test code contains two sections:

1. Data Recording Code (DRC): The correctness of simulation program requires that all model state data is correct. Therefore, test code should be able to save all model run-time data, including the initialization, input and output data of models, into files or database to facilitate testers to verify whether there are any errors or not.
2. Test Case Execution Code (TCEC): Given a test case, TCEC can be generated by PDES-TCG automatically according to this case, used to verify whether the simulation results are in line with expectations.

The paper is organized as follows. The second section describes RUM. The third section is related work. The fourth section introduces PDES-TCG. The next 5 sections are used to describe the five phases of PDES-TCG in detail. The last section concludes the paper work.

RUM

Yao and Zhu et al. [3] proposed a reusable model development specification - RUM, as shown in figure 1, which defines the interface of model initialization, input data and output data etc. RUM can improve PDES application development efficiency and provide basis for automatic testing. The specification has been applied in a number of projects and achieved good results.

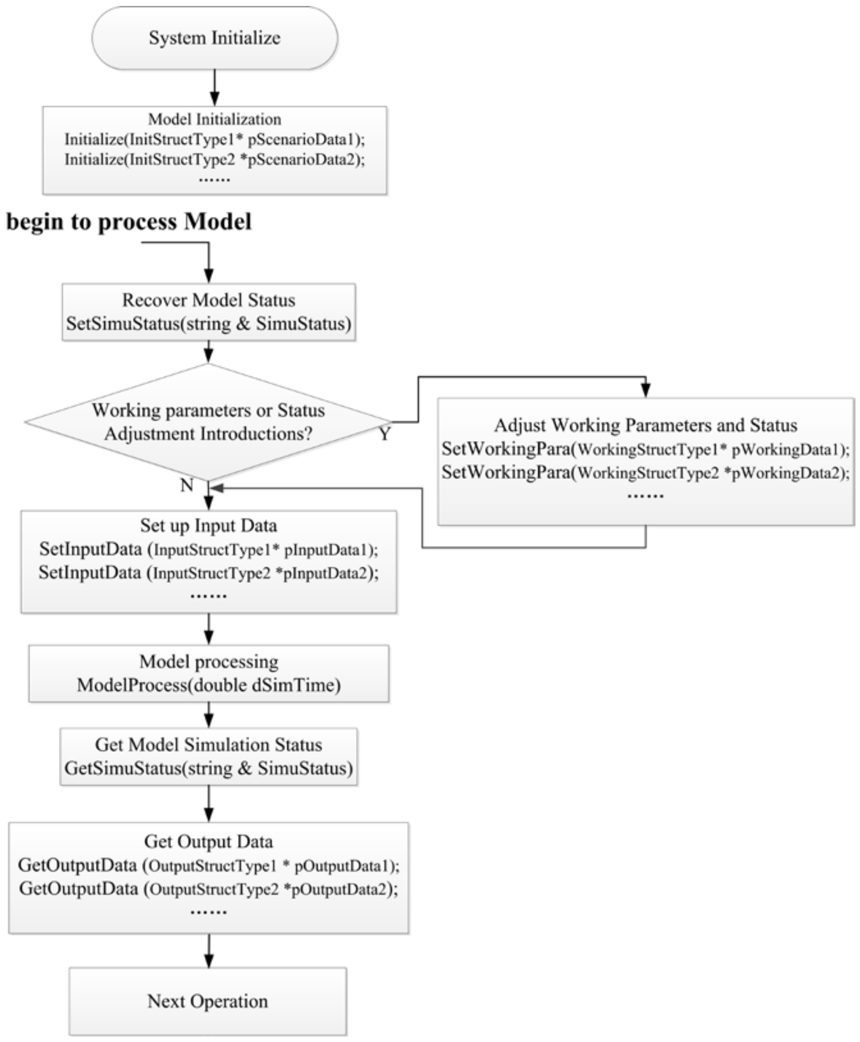

System Initialize
Model Initialization
Initialize(InitStructType1* pScenarioData1);
Initialize(InitStructType2 *pScenarioData2);
......
begin to process Model
Recover Model Status
SetSimuStatus(string & SimuStatus)
Working parameters or Status Adjustment Introductions?
Y
N
Adjust Working Parameters and Status
SetWorkingPara(WorkingStructType1* pWorkingData1);
SetWorkingPara(WorkingStructType2 *pWorkingData2);
......
Set up Input Data
SetInputData (InputStructType1* pInputData1);
SetInputData (InputStructType2 *pInputData2);
......
Model processing
ModelProcess(double dSimTime)
Get Model Simulation Status
GetSimuStatus(string & SimuStatus)
Get Output Data
GetOutputData (OutputStructType1 * pOutputData1);
GetOutputData (OutputStructType2 *pOutputData2);
......
Next Operation

Fig.1. RUM described in C++ language

Related Work

After years of development, program testing technology has been mature, but little research is special for PDES applications.

There are some parallel discrete event simulation platforms, such as SPEEDS (Synchronous Parallel Environment for Emulation and Discrete Event Simulation)[4][5]、GTW(Georgia tech Time Warp)[6][7]、 YHSUPE (YinHe Simulation Utilities for Parallel Environment)[8] etc. These platforms provide efficient runtime environments for PDES applications, but depending on manual test.

Xu[9] proposed a tool, ISTA (Integration and System Test Automation), for automated generation of executable test code by using high-level Petri nets as finite state test models. The input to ISTA should follow MID specification, consisting of a Predicate/Transition net, a MIM (model-implementation mapping), and HC (helper code). However it would cost a lot to achieve MID for PDES application.

FTS [10], FedProxy [11] and other HLA federate testing tools have been very mature. However, the coupling degree among PDES entities is closer than that among HLA federates, and the operation object should be recognized when an event is scheduled. Besides, there is no specification of information interaction methods and types among simulation entities. So current federate testing techniques cannot be used to test simulation entities directly and cannot be used for PDES application test either.

PDES-TCG

PDES-TCG can be described as consisting of five phases. The first phase is to obtain the custom data structures in models. The second phase is to obtain model parameters in simulation entities, and the models should be developed following RUM. These parameters consist of model initialization data parameters, input data parameters and output data parameters. The third phase is to generate data recording code and insert them into application programs. The fourth phase is to extract test data from given test case XML file. The fifth phase is to generate test case execution code.

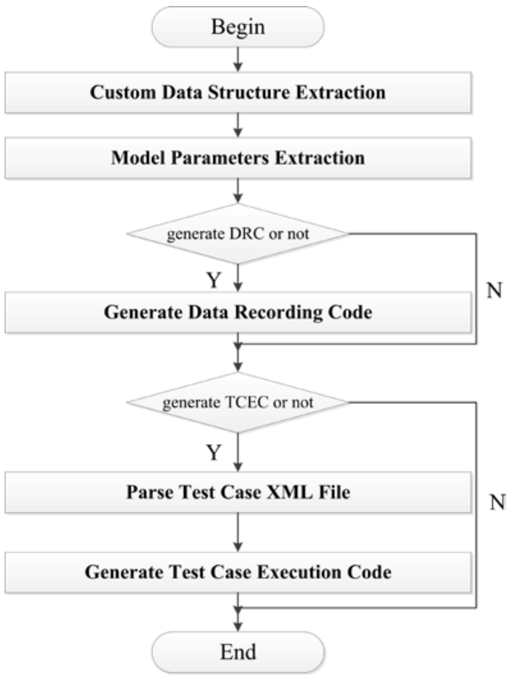

Begin
Custom Data Structure Extraction
Model Parameters Extraction
generate DRC or not
Y
N
Generate Data Recording Code
generate TCEC or not
Y
N
Parse Test Case XML File
Generate Test Case Execution Code
End

Fig.2. PDES-TCG flow chart

Custom Data Structure Extraction

Models in PDES application usually contain some custom data structures. This phase is to obtain all custom data structures in models. Here a special CA (code analyzer) should be developed according to model code specification. By scanning model code files, CA can extract these data structures automatically. For simulation models developed by C ++ language, custom data structures are declared in header files in general, so CA should scan the header files.

CA mainly consists of a lexical analyzer and a parser, used to locate custom data structure declaration sections. If models are developed by C ++ language, Lexical analyzer is a C ++ lexical analyzer. Here the parser is equivalent to a lightweight C ++ parser. Its grammar rules are simple, only containing data structure declaration rules, such as struct declaration in C ++. When extracted, custom data structures will be stored in a string vector *S*, including data structure name, names and types of internal members, taking data structure name as index.

Model Parameters Extraction

PDES application consists of simulation entities, entity contains event. Model is scheduled in event, so model instance and model parameters are declared in simulation entity. This phase is to extract model data parameters automatically by scanning the simulation entity code files. Here the CA mentioned before need to be reused, adding parameter declaration rules into the grammar rules of parser. When identifying a parameter, CA can determine whether it is a model parameter by querying its data structure name in vector *S*. If the parameter belongs to model data parameters, its name and data structure name are stored in string vector *V*, taking parameter name as index.

When simulation models are developed following RUM, it can be directly discriminated that which parameters corresponding to model initialization data parameters, input data parameters or output data parameters. Therefore, RUM specification is great helpful for automatic generation of test code.

Generate Data Recording Code

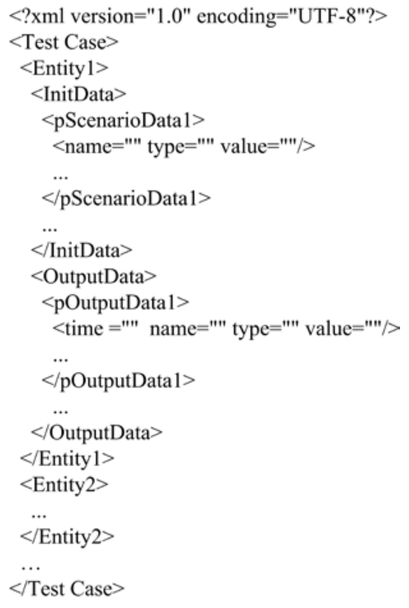
Model data recording code is used to record the values of model parameters and save them into files during program execution, including initialization, input and output data. By examining these files, testers can check out whether simulation result is true or not. Each parameter corresponds to a TXT file or a database table.

For simulation objects developed in C ++ language, data recording code consist of two parts, declaration part and execution part. If we want save data into files, so declaration part should include file pointers and file operation functions declaration. One file pointer corresponds to one parameter. File operation functions are used to write the values of parameters into files according to its data structure. File operation functions can be used as methods of simulation entities. Execution part is to open a TXT file with a file pointer and write values into files by calling file operation functions.

Parse Test Case XML File

Test cases are commonly used to test application program. When PDES application has integrated all simulation entities, the input field is the initialization data for all simulation entities. Given that XML file has a clear hierarchy, using XML file to store test case makes test data extraction easy. The internal structure of test case XML file is shown in figure 3. Test case contains all the initialization and expected output data of each entity. The same event in one entity may be scheduled multiple times at different simulation time during the simulation runtime, so the entity will generate output data repeatedly. Simulation time must be considered when comparing expected output data with the actual output data of entities. That is why the output data in test case files should be bound to simulation time.

According to the XML file structure, we can design an XML parser to extract the test data of each entity and save them into a string vector *T*, taking entity number as index.



```
<?xml version="1.0" encoding="UTF-8"?>
<Test Case>
  <Entity1>
    <InitData>
      <pScenarioData1>
        <name="" type="" value=""/>
        ...
      </pScenarioData1>
      ...
    </InitData>
    <OutputData>
      <pOutputData1>
        <time =""  name="" type="" value=""/>
        ...
      </pOutputData1>
        ...
    </OutputData>
  </Entity1>
  <Entity2>
    ...
  </Entity2>
  …
</Test Case>
```

Fig.3. Test case template based on RUM

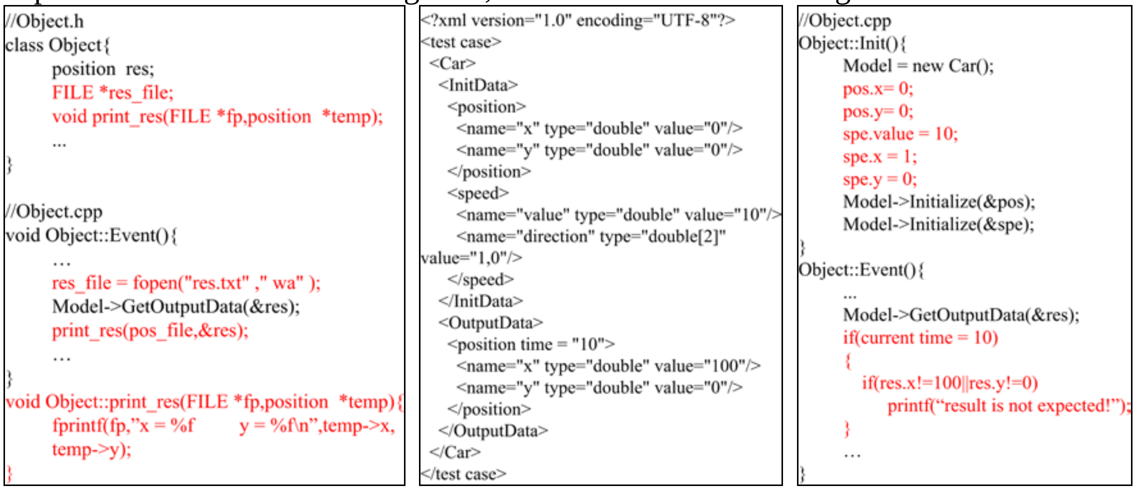
Generate Test Case Execution Code

When using a test case to test application, it will cost a lot to add the assignment code and comparison code into the simulation entity code manually. Generating code automatically will be an effective solution. According to RUM specification, Initialize function is used to transfer initialization data to model instance by parameters, while GetOutputData function transfers the output data of model to model output parameters.

When scanning to Initialization function, data structure name can be obtained by querying the parameter name in vector *V*, as well as data structure detail information by querying vector *S*. According to the entity number, test data can be obtained from vector *T*. All these information can be used to generate test case execution code automatically. Code can be divided into two parts, initialization assignment code and output data comparison code. The former is responsible for assigning the initialization data extracted from test case to the model initialization parameters. The latter compares the expected output data with model output parameters.

Validation

A simple PDES application was accomplished on parallel discrete event simulation engine YHSUPE. The application contains a simulation entity, the entity contains a self-scheduled event, and a Car model instance is called in the event. For the model, the initial data is starting position and velocity, output data is the latest position. The total simulation time was 10, and the event was scheduled once every 1 simulation time, then the position data of car model instance was updated per simulation time. Based on PDES-TCG, we achieved a tool used to generate test code for PDES application. With this tool, data recording code can be generated directly. Besides a test case is given, containing the initial data and expected output data, test case execution code can also be generated. Experiment results show that test code can correctly record output data of the model, as well as determine whether the actual output data is in line with the expected output data from test case in specified time. As shown in figure 4, the red code section belongs to test code.



```
//Object.h
class Object{
        position  res;
        FILE *res_file;
        void print_res(FILE *fp,position  *temp);
        ...
}

//Object.cpp
void Object::Event(){
        …
        res_file = fopen("res.txt" ," wa" );
        Model->GetOutputData(&res);
        print_res(pos_file,&res);
        …
}
void Object::print_res(FILE *fp,position  *temp){
        fprintf(fp,”x = %f        y = %f\n”,temp->x,
        temp->y);
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<test case>
  <Car>
    <InitData>
      <position>
        <name="x" type="double" value="0"/>
        <name="y" type="double" value="0"/>
      </position>
      <speed>
        <name="value" type="double" value="10"/>
        <name="direction" type="double[2]"
value="1,0"/>
      </speed>
    </InitData>
    <OutputData>
      <position time = "10">
        <name="x" type="double" value="100"/>
        <name="y" type="double" value="0"/>
      </position>
    </OutputData>
  </Car>
</test case>
```

```
//Object.cpp
Object::Init(){
        Model = new Car();
        pos.x= 0;
        pos.y= 0;
        spe.value = 10;
        spe.x = 1;
        spe.y = 0;
        Model->Initialize(&pos);
        Model->Initialize(&spe);
}
Object::Event(){
        ...
        Model->GetOutputData(&res);
        if(current time = 10)
        {
            if(res.x!=100||res.y!=0)
                printf(“result is not expected!”);
        }
        …
}
```

(a) data recording code (b) test case example (c) test case execution code

Fig.4. experiment results

Conclusion

This paper proposes a method PDES-TCG for PDES application program testing. First, the importance of automatic test code generation for PDES application was summarized. Second, we listed the limitations of existing test code generation tools and methods when used for PDES application, and concluded that existing test code generation methods could not apply to PDES application directly. Next, we proposed a method PDES-TCG based on RUM, which can be used to generate executable test code and insert them into application programs automatically. The test code can record run-time data of simulation models and test simulation entities with given test cases. At last, PDES-TCG was validated with a simple PDES application. We achieved a tool base on PDES-TCG, which generated test code for the application correctly and inserted them into program files. Given the current testing status of PDES application test, most of the work is done manually. Therefore, this method has great practicability and can significantly improve test efficiency in some extent.

Acknowledgement

We appreciate the support from State Key Laboratory of High Performance Computing, National University of Defense Technology (No.201303-05) and Research Found for the Doctoral Program of Higher Education of China (No.20124307110017).

References

[1] Richard M. Fujimoto. Parallel and Distributed Simulation Systems. USA: John Wiley&Sons, Inc.2000.

[2] A. van Deursen, L. Moonen, A. van den Bergh, and G. Kok. Refactoring test code. In M. Marchesi and G. Succi, editors, Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001), pages 92-95, May 2001.

[3] Zhu F, Yao Y P, Chen H L and Yao F. Reusable Component Model Development Approach for Parallel and Distributed Simulation [J]. The Scientific World Journal, March, 2014.

[4] Steinman Jeff, SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation, in proceedings of Advances in Parallel and Distributed Simulation, Pages 95-103, 1991.

[5] Bailey Chris, McGraw Robert, Steinman Jeff, and Wong Jennifer, SPEEDES: A Brief Overview, In proceedings of SPIE, Enabling Technologies for Simulation Science V, Pages 190-201, 2001.

[6] Kalyan S. Perumalla and Richard. M. Fujimoto, GTW++ An Object oriented Interface in C++ to the Georgia Tech Time Warp System, Georgia Institute of Technology, 1996.

[7] Richard. M. Fujimoto, S. R. Das, and K. S. Panesar, Georgia Tech Time Warp programmer’s manual, Technical report, College of Computing, Georgia Institute of Technology, Atlanta, GA, july 1994.

[8] YAO Yi-ping, ZHANG Ying-xing, Solution for Analytic Simulation Based on Parallel Processing, journal of system simulation, vol.20, no.24 , Pages 6617-6621, 2008.

[9] A Tool for Automated Test Code Generation from High-Level Petri Nets. Dianxiang Xu. 32nd International Conference, PETRI NETS 2011 Proceedings.

[10] Frank M. Valdes, Kevin Mullally, Jaime Cisneros, David W. Roberts, Margaret Horst, J. Andrew Old, Rodney Long. Lessons Learned from Design and Development of the Federation Test System (FTS). 2004:102-111.

[11] Horst M, Old J A. Federation testing process and tools: federation test and management integration supportion support and study final results report[R]. Georgia Tech Research Institute, 1998.