The Design and Implementation of Workflow Engine for Spacecraft Automatic Testing

Gang Ye
State Key Laboratory of Software Development Environment, Beihang University, Beijing, China
Email: yegang@nlsde.buaa.edu.cn

Xianjun Li, Dan Yu, Zhongwen Li, Jie Yin
State Key Laboratory of Software Development Environment, Beihang University, Beijing, China
Email: {lixianjun, yudan, lizhongw, yinjie}@nlsde.buaa.edu.cn

Abstract—Spacecraft software automatic testing plays an essential role in spacecraft testing field for a decade. However, less attention has been paid on it based on workflow methodology. In this paper we introduce workflow technology into testing for improving the automation of spacecraft software testing. Firstly we present a spacecraft testing process definition language STPDL. Then workflow patterns framework is proposed for analyzing the language. In addition, we implement a core scheduling algorithm in our workflow engine and analyze its running time. At the end of the paper we evaluate the effect and efficiency of the testing workflow engine.

Index Terms—spacecraft software testing; automatic testing; workflow engine

I. INTRODUCTION

Spacecraft software automatic testing plays an essential role in spacecraft testing field for a decade. However, in China less attention has been paid on this field until two or three years ago. Nowadays, some testing missions are still completed by testers manually, which gives birth to a great number of artificial mistakes. In addition, the complex requirements for spacecraft software testing lead to a lot of workload, as well as the difficulties of testing plan, implementation and management [1, 2, 3]. We summary all these situations as follows:

a) Testers have to check intermediate results one by one through testing.

b) Parallel testing is almost unavailable in manual.

c) Testers have to prepare testing data again for another testing plan, especially in regression testing.

In our opinion, these problems are resulted from two aspects:

a) Lack of testing criterion. Currently, test cases are written by experienced testers or test-teams. Different writing manners maybe result in different test cases even with the same testing.

b) Lack of automatic testing execution and supervision mechanisms. Because the supervision of testing process and the judgement of intermediate results are completed by testers manually, it’s difficult to finish an operation which needs to inspect a group of parameters in a few seconds, even for an experienced tester.

In order to solve these problems, in this paper we aim to provide a spacecraft testing process definition language STPDL as a standard for establishing testing cases, and implement a spacecraft software automatic testing workflow engine (SSATWE) which can be used to execute and monitor testing processes automatically.

The paper is structured as follows. Section 2 gives an overview of the related work in the business process description language, and analyzes the state-of-the-art in workflow management systems. Section 3 proposes a spacecraft testing process definition language STPDL and evaluates it with workflow patterns framework. Section 4 presents a core scheduling algorithm in our workflow engine. Section 5 compares the efficiency of scheduling algorithm with those in jBPM and Shark. Section 6 concludes the paper and gives the future of our work.

II. RELATED WORK

A. Spacecraft software testing

The same as any engineering project, the spacecraft in the development stage can not be separated from a large number of trials from the beginning to the end. In order to determine whether the spacecraft performance and functionality can meet the design requirements or not, various indicators of the spacecraft need to be tested. Here the spacecraft software testing refers to using various software and hardware systems to test each computer subsystem in the spacecraft, detecting if the work in internal subsystem or between the subsystems of each spacecraft is in right way.

Spacecraft software testing has the following characteristics that testing process is complicate and changeable, the testing requires highly real-time response, and each process of the testing is similar. At present
spacecraft testing still depends on manual work to a large extent, thus resulting in the low efficiency considering the above testing characteristics. So workflow technology, whose main function is to realize the automation of business process, can help to solve this problem. But currently few researches introduce workflow into the spacecraft software testing field.

B. Business process description language

Traditionally, IT standards have been proposed as a basis for achieving increased interoperability between distinct technological offerings. Since processes play a significant role in the functioning of BPMs, it is indispensable to use an established language as standard for describing and modeling business processes [4, 5]. XPDL and BPEL are typically business process description languages in this field.

XPDL [6] uses XML to describe a business process, which is based on the recognized workflow standard, WfMC reference model. XPDL is conceived of as a graph-structured language with additional concepts to handle blocks. Scoping issues are relevant at the package and process levels.

BPEL [7] is an XML based flow languages that provides a formal specification of business process behavior based exclusively on web services, whose definition is made up of four basic components: activities, control links, a series of partner-links and variables.

However in recent years, many standards initiatives seem increasingly to hamper the entry of new solutions providers. The standards are both informally defined and subject to ongoing revision (e.g. XPDL v1.0, BPEL v1.1) meaning developers continually need to update their software (sometimes markedly) in order to continue to be compliant as the standard(s) evolve [8]. With this in mind, whilst being mindful of industry standards, we should not view them as a panacea which will necessarily increase the usage and relevance of their offering. In addition, due to the special characters of spacecraft, these representative approaches may not be directly used in modeling spacecraft software testing.

C. Business process management

Business process management systems (BPMs) are software systems which support the execution of business processes in an organization. They take care of the distribution of work to the right resources at the right time. With more than twenty years development [9], there are a wide variety of workflow and business process management offerings available, open source and closed source. E.g. jBPM, Enhydra Shark, and WebSphere MQ are three of the most famous BPMs, while the first two are open source systems and the rest is a closed source one.

jBPM is a JBoss workflow and business process management engine in Java that can execute processes described in BPEL or its own process definition language jPDL. The business process model of jBPM is based on activity diagram, and the construction of its engine considers the theories of FSM and Petri-Net. jBPM has a process definition tool and a workflow engine called JBoss JBPM which takes care of the execution of process instances [8].

Enhydra Shark is a Java workflow engine offering from Together Teamlösungen and ObjectWeb [10]. It is an extendable and embeddable Java Open Source workflow engine framework including a standard implementation completely based on WfMC’s specifications using XPDL (without any proprietary extensions) as its native workflow process definition format and the WfMC “ToolAgents” API for server-side execution of system activities. In Enhydra Shark, the workflow engine is called Shark which takes care of the execution of process instances [10].

IBM WebSphere MQ is a family of network communication software products launched by IBM. WebSphere MQ is IBM's Message Oriented Middleware offering [11]. Like other commercial offerings, maybe it will spend a lot of cost to rebuild spacecraft testing environment and deploy the product into. Another problem is coming from embargoed, enterprises like IBM maybe can’t sell such offering to our country.

III. SPACECRAFT TESTING PROCESS MODELING

After analyzing spacecraft software automatic testing demands, we find that workflow languages like XPDL and BPEL couldn’t satisfy testing requirements, especially in the way of safety testing process description and modeling capability. Some control-flow patterns in these languages may bring in unsafe testing process description, while some other control-flow required by spacecraft software testing are not supported by them or their implements — JBPM or Shark.

In order to model testing process correctly, a testing process description language, which specifies a criterion to indicate the start and termination of a process and the relationships of all testing activities in a process, should be provided. Testers use this language to define specific testing processes which used by engines to perform.

On the basis of workflow technology, we propose a spacecraft testing workflow process definition language STPDL to construct the workflow model of spacecraft software automatic testing, and implement the required and adequate flow patterns [12].
A. Definition of basic elements of STPDL

Before presenting the language, we give the definition of some basic language elements.

Definition 1 (Basic elements definition of STPDL [1]):

- **Test Procedure Design**: a complete test procedure design method.
- **Test Procedure**: a combination of test processes which are correlated logically.
- **Test Resources**: all resources used in test procedure, including test device resources, test data resources and test atom resources.
- **Test Process**: test function that complete a test, including test sub-process and test atom activity.
- **Test Sub-Process**: a special function operation in test process, for example, spacecraft power-on sub-process. It comprises a combination of test atom activities and test transitions.
- **Test Atom Activity**: a basic and indivisible test operation in a test process, for example, sending an instruction to spacecraft.
- **Test Transition**: testing control-flow transition information between related activities.
- **Test Device Resources**: all services on testing equipments and software in spacecraft testing.
- **Test Data Resources**: all testing data in testing which comes from databases or file servers, including test preparation data and test result data.
- **Test Atom Resources**: all atom entities invoked by test atom activity to archive atomic operations in a test process.

According to testing demands, we define STPDL to standardize the testing process, which is based on XML in view of scalability and flexibility. Compared with XPDL [6], STPDL describes testing processes by using hierarchy. In our language, we use atomic testing activities, “Test Atom Activity”, as basic testing units, which perform test actions by using test data resources, accessing test devices resources and invoking test atoms resources. A test sub-process is a set of some logical related test atom activities, which accomplishes a special operation. A test process is used to encapsulate test sub-processes and test atom activities. A test procedure is a combination of test processes, with which test processes can be executed parallel.

B. Spacecraft testing process definition language

The presented language STPDL provides a formal definition to model the spacecraft testing process. Figure 1 gives the STPDL meta-model, which describes the relationships between the entities of the top level.

In Definition 2, we use BNF representation to In Definition 2, we use BNF representation to describe this language, where ::= means “is defined as”, [ ] represents the including elements is optional, <> represents the including elements is required, {} represents the including elements occur zero or more times, and | means “or”.

Definition 2 (Definition of spacecraft testing workflow process description language [1])
TABLE I.
SUPPORT FOR THE CONTROL-FLOW PATTERNS IN A–SSATWE AND B–ENHYDRA SHARK 2.0

<table>
<thead>
<tr>
<th>Basic Control-flow</th>
<th>A</th>
<th>B</th>
<th>Termination</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Sequence</td>
<td>+</td>
<td>+</td>
<td>11. Implicit Termination</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>2. Parallel Split</td>
<td>+</td>
<td>+</td>
<td>43. Explicit-Termination</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>3. Synchronization</td>
<td>+</td>
<td>+</td>
<td>Multiple Instances</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Exclusive Choice</td>
<td>+</td>
<td>+</td>
<td>12. MI without Synchronization</td>
<td>–</td>
<td>+</td>
</tr>
<tr>
<td>5. Simple Merge</td>
<td>+</td>
<td>+</td>
<td>13. MI with a pri. Design Time Knl.</td>
<td>+</td>
<td>–</td>
</tr>
<tr>
<td>Advanced Synchronization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Multiple Choice</td>
<td>+/–</td>
<td>+</td>
<td>14. MI with a pri. Runtime Knl.</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>8. Multiple Merge</td>
<td>–</td>
<td>–</td>
<td>34. Static Partial Join for MI</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>10. Arbitrary Cycles</td>
<td>+</td>
<td>–</td>
<td>15. MI without a pri. Runtime Knl.</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>12. MI without Synchronization</td>
<td>–</td>
<td>+</td>
<td>36. Dynamic Partial Join for MI</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>13. MI with a pri. Design Time Knl.</td>
<td>–</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14. MI with a pri. Runtime Knl.</td>
<td>–</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15. MI without a pri. Runtime Knl.</td>
<td>–</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State-Based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18. Milestone</td>
<td>–</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19. Cancel Activity</td>
<td>–</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20. Cancel Case</td>
<td>+/–</td>
<td>+</td>
<td>41. Thread Merge</td>
<td>+/–</td>
<td>–</td>
</tr>
<tr>
<td>21. Structured Loop</td>
<td>–</td>
<td>–</td>
<td>42. Thread Split</td>
<td>+/–</td>
<td>–</td>
</tr>
<tr>
<td>22. Recursion</td>
<td>–</td>
<td>+</td>
<td>25. Cancel Region</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>23. Transient Trigger</td>
<td>+/–</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24. Persistent Trigger</td>
<td>–</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In our opinion, STPDL mainly focuses on spacecraft testing process, while XPDL, BPEL and etc. are dedicated to general business process. With the requirement of actual testing, we should avoid dangerous control-flow patterns which maybe supported by general business process, e.g. Local Synchronizing Merge in jBPM and Arbitrary Cycles in Shark [8]. STPDL directly avoids such control-flow patterns by no-support.

IV. SPACECRAFT SOFTWARE AUTOMATIC TESTING WORKFLOW ENGINE

A. Architecture of spacecraft testing workflow engine

We design the engine with three layers: user interface layer, workflow engine core layer and resource layer. Figure 2 shows the architecture of the engine.

The user interface layer is composed of four parts: test procedure interface, test procedure result interface, engine workflow control interface and execution information display interface. It provides with interaction information between the testers and the engine. By test procedure interface, the engine accepts test procedure for executing.

The workflow engine core layer is in charge of the execution of workflows, which is designed to parse and execute test procedures defined with STPDL.

Figure 2 shows the architecture detail of the engine. With the hierarchical structure, it is more convenient to distinguish the core module from other parts and make the main function of engine low coupling and high cohesion.

The resource layer stores three predefined resource bases: test data resource, test device resource and test atom resource. It stores this information in their native XML format, because all the data that are manipulated by the software are either XML-based, or encapsulated in XML.

B. Core scheduling algorithm for atom activities instantiation

There are two types of execution mode in testing: sequence or parallel, where we treat if-else construct and any cycle construct as sequence mode. As the schedule approaches of different engines are close in sequence execution, the efficiencies of BPMs are almost the same. In
parallel mode, SSATWE utilizes thread-pool to create and execute each activity instance respectively, while in jBPM and Shark new instances of tasks are spawned off through the use of a loop. The different designs between engines are resulting from different application backgrounds. In spacecraft real-time testing, data frames are broadcasted one after another on network every 500 milliseconds. In our experiments, it takes about 44 ms to 47 ms for instantiating a test atom activity averagely. Thereby, suppose that a testing is starting at the same time with a data frame \( f_i \) feasible on network. Theoretically with the scheduling approach in jBPM (or Shark), for a testing process includes more than ten parallel activities, if all activities want to fetch data from data frame \( f_i \), then the last activity can never finish its task. Because when it is created, the data frame is covered by the next.

Emphasizing computational simplicity and data access, a basic model for average execution cost for parallel instantiation operation, \( C_{avg} \), is used. It includes four parts: activity context initialization time, activity instantiation time, test activity resource locating time, and data access time:

\[
C_{avg} = \frac{\sum_{i=1}^{k} (C_{cont} + C_{ins} + C_{res} + C_{data})}{k}
\]

which can be decomposed as:

\[
C_{cont} = \sum_{i=1}^{n} \tau_{cont}
\]

\[
C_{ins} = \sum_{i=1}^{n} \tau_{ins}
\]

\[
C_{res} = \sum_{i=1}^{n} p_{r/w}^{res} m_{r/w}^{res} t_{res}
\]

\[
C_{data} = \sum_{i=1}^{n} p_{r/w}^{data} m_{r/w}^{data} t_{data}
\]

where \( k \) is experimentation count; \( n \) is parallel activity count; \( \tau_{cont} \) is the time for initializing context for an activity instance; \( \tau_{ins} \) is the time for creating an activity instance; \( m \) is number resources invoked; \( l \) is number data accessed and \( t \) is the access time for a specific testing resource to testing atom activity. The superscript of a variable indicates the resource type (res stands for test atom activity resource, while data stands for test data resource), and the subscript indicates a read or write access. Each resource access cost exists for an individual instantiation with some probability, \( p \), which represents a resource miss for a read or the likelihood that a write operation cannot be posted. As the objective of our evaluation is aimed at the efficiency and effectiveness of two instantiation approaches, we store all of the test atom activities in local machine and set probability of
reading or writing to 1 (namely \( p = 1 \)), which will make \( \bar{C} \) to be maximum in the same condition and exclude resource-no-found factor.

V. E VALUATIONS

The performance scalability relationships are reflected in Figure 3, which displays a detailed breakdown of the time taken for varied amount activity instantiation by Shark or jBPM pattern and SSATWE. Every test scenario is designed in the following pattern: in every experiment, we run a test progress contains \( n \) activities, where \( n \) is an even number arranged from 2 to 22. In these \( n \) activities, we let the first half activities run in sequence mode, and the rest run in parallel mode. Meanwhile we record three types of time data for parallel activities related to the last serial activity’s finish time: MEIT (Most Earliest Instantiation Time), AIT (Average Instantiation Time) and MLIT (Most Latest Instantiation Time). In the chart, axis x describes the number of parallel activities, while axis y describes the MLIT time data. From the chart, we can see that if the number of parallel activities is greater than eleven, then the time spent for instantiations in Shark will be greater than 500ms, which means the 11th activity will miss the data frame \( i \) and cause logical error in testing. The chart also shows that our implementation reveals a less time consumption for instantiating the same number of activities, while in Shark, time consumption increases sharply.

We carry out another experiment to check by SSATWE how many parallel activities can be instantiated together. Figure 4 and Table II show our testing results. From the result, we can see that when the number of parallel activities is bigger than 500, the MEIT will greater than 500ms. In this test scenario, we find that the time-consumption is related to the situation of the computer, especially the CPU-usage and memory-consumption. By our experience, the number of parallel activities is better not greater than 400, and the computer is not used for high CPU-usage and memory-consumption application.

VI. C ONCLUSIONS

Spacecraft software testing is a matter of growing importance. However, because the testing efficiency is very low due to the complex testing flow, it becomes a tedious task. Our approach exploits workflow technology to characterize spacecraft testing flow so that the automatic level of spacecraft testing will be improved. In this paper we propose a workflow description language—STPDL. Based on this language, we design and implement a workflow engine. At the same time, we give the evaluations of STPDL control-flow patterns and the scheduling effect of the engine. Now, the system has already applied in actual testing work.

As the future work we will add workflow verification and evaluation into execution implementation, and provide a better visual modeling environment to the spacecraft testers.

REFERENCES


Ye Gang (1982 - ) received the BS degree in computer science from Zhejiang University of Technology, China, in 2005. He is currently a PhD candidate in State Key Laboratory of Software Development Environment, Beihang University, Beijing, China. He is a student member of the ACM and also a student member of the CCF.

His research interests include spacecraft automatic testing, spacecraft integrate test, software testing, fault localization.

Li Xianjun (1977 - ) is currently a PhD candidate in State Key Laboratory of Software Development Environment, Beihang University, Beijing, China.

His research interests include spacecraft automatic testing, spacecraft integrate test information Processing, component technology.

Yu Dan (1979 - ) received the Ph.D in computer science from Wuhan University, China, in 2007 and then become a postdoctoral in Beihang University. She is a member of the IEEE and also a member of the CCF. Now she is a lecturer in the National Key Lab for Software Development Environment of the school of computer science and engineering in Beihang University.

Her research interests include spacecraft test language, safety critical system, and software automatic testing.

Li Zhongwen (1981 - ) received the BS degree in computer science from Beihang University, China, in 2004, and the MS degree in computer science from Beihang University, China, in 2006. He is currently a PhD candidate in State Key Laboratory of Software Development Environment, Beihang University, Beijing, China.

His research interests include workflow, spacecraft automatic testing, and real-time system.

Yin Jie (1984 - ) received the BS degree in Mathematics from Beihang University China, in 2006. He is currently a PhD candidate in State Key Laboratory of Software Development Environment, Beihang University, Beijing, China. He is a student member of the ACM and also a student member of the CCF.

His research interests include program verification, software automatic testing, and programming languages theory.