日志分析法在一卡通系统测试中的应用*

齐 林, 郭 陟, 顾 明
(清华大学 软件学院, 北京 100084)

摘 要: 针对一卡通系统测试中因输出结果数量大、系统关联复杂、测试准则构造困难等因素而导致的难以判断输出结果正确性的问题,提出了扩展日志分析法。介绍了实现该方法的系统结构和所使用的测试状态机,引入了一种自行开发的支持层次化状态机、具有执行自定义函数功能的状态机定义语言,增强了扩展日志分析法测试复杂大型系统的能力。该方法在一卡通系统测试中得到了应用。

关键词: 软件测试; 日志分析法; 状态机; 状态机定义语言
中图法分类号: TP311.5 **文献标识码**: A **文章编号**: 1001-3695(2006)05-0118-03

Using Log File Analysis in One Card Multi-service System Testing

QI Lin, GUO Zhi, GU Ming
(School of Software, Tsinghua University, Beijing 100084, China)

Abstract: To solve the problem of checking testing output due to the large amount of the output and the difficulty in constructing the test oracle in One Card Multi-service System Testing, proposes an extended Log File Analysis method. Introduces the architecture of the system implementing the method and the Testing State Machine involved. Describes a new state machine definition language which supports multi-layered state machine, has the ability of invoking customized function and increase the applicability of the extended Log File Analysis in testing large complex systems. The method is applied in One Card Multi-service System Testing.

Key words: Software Testing; Log File Analysis; State Machine; Testing State Machine Language

市政交通一卡通系统 OCMS(One Card Multi-service System,以下简称“一卡通系统”)是一套与人们日常生活关系密切的电子交易系统。该系统要求较高的安全性和可靠性,需要进行严格的测试。测试中遇到了输出结果数据量大、系统关联复杂、测试准则(Test Oracle)构造困难等问题,难以判断输出结果的正确性。手工测试方法效率低、易出错;使用基于测试准则的方法如回归测试法[1]、断言测试法[2]等也具有一定的困难。日志分析法(Log File Analysis, LFA)[3]使用状态机分析被测系统的日志实现软件测试,具有构造简单、描述能力强等特点,较适合应用于关联复杂的系统测试中。本文针对一卡通系统的特点,提出了支持层次化状态机的扩展日志分析法,增加了执行自定义函数的功能,使之更适用于一卡通系统的测试,并基于该方法实现了测试系统,在一卡通系统测试中取得了良好的效果。

1 研究背景

1.1 一卡通系统

一卡通系统是一个小金额电子交易系统,由消费 IC 卡、消费机具(出租车计价器、公交刷卡机等)、客户端(出租客户端,公交客户端等)、各级分中心、总中心以及数据传输网络等组成。消费者在进行交易时,消费机具对消费 IC 卡进行读写,产生交易数据。交易数据定期从消费机具中采集到客户端中,通过计算机网络逐级上传,最终汇总到总中心,进行结算划账。

一卡通系统内部关联复杂,对可靠性和安全性的要求较高,中间输出和最终输出与前期的运行情况关系紧密、变化大。这些特点要求系统在开发、试运行以及升级维护时,都需要进行大量的测试,要求使用的测试方法效率高,能适应复杂系统和输出结果与前期关系紧密的情况。

1.2 日志分析法

日志分析法是文献[3]中提出的一种用状态机对被测系统的日志进行分析实现软件测试的方法。该方法要求被测系统产生文本日志,记录测试过程中的输入、输出、函数调用以及返回值等重要事件。日志被输入状态机进行分析,实现测试。状态机由测试人员根据被测系统的需求转换生成。当状态机在日志的驱动下终止于终态时,说明日志符合状态机的要求,即被测系统符合需求;否则,不符合要求。

日志分析法与回归测试法、断言测试法等相比,具有效率高、构造简单、依据状态机判断测试输出的正确性等特点。但文献[3]中定义状态机的 LFAL 语言描述能力受限:对状态机不能进行层次化的设计,不宜描述大型系统;没有执行自定义函数等功能,对复杂系统描述困难。

2 基于日志分析法的一卡通测试系统

2.1 总体结构

基于日志分析法并针对一卡通系统的特点而提出了扩展

收稿日期: 2005-04-22; 修返日期: 2005-06-20
基金项目: 国家“863”计划项目(2003AA414031, 2003AA413031)

日志分析法, 同时实现了基于该方法的一卡通日志分析测试系统——OLFTS(OCMS Log File Testing System) 。系统总体结构如图 1 所示。

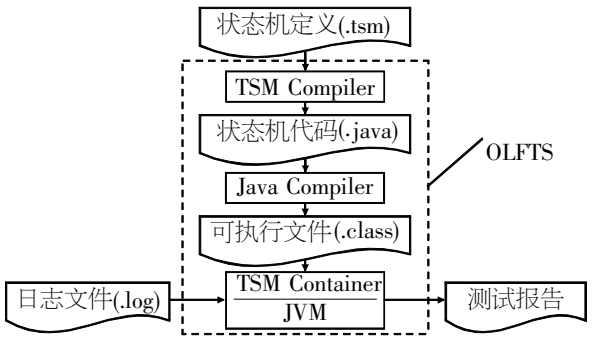

图 1　OLFTS 总体结构

首先, 将一卡通系统的一个或一组相近的需求转换为由函数调用、消息发送等描述的执行过程, 并确定函数调用等发生的条件以及返回值等。然后, 用状态机定义语言 TSML(Testing State Machine Language) 定义一个描述该过程的状态机(.tsm 文件) , 并输入状态机编译器, 将其编译为某种语言源代码(以 Java 语言为例) , 作为中间结果。调用相应语言的编译器将中间结果编译为可执行文件, 最终得到可执行的状态机。测试时, 先将状态机可执行文件输入状态机容器中, 再将被测系统的日志文件输入状态机容器。状态机容器负责读取日志、创建状态机, 广播日志, 收集状态机输出和产生测试报告等工作。状态机在日志驱动下进行状态转移, 并向容器输出信息。

2.2　日志

测试需要的日志由被测系统产生。每一个日志行表示一个事件的发生, 例如函数调用、消息发送等。日志行由行单元组成, 行单元之间用空格分隔。每个日志行开头的行单元称为关键行单元, 必须是一个字符串, 通常是被调用函数的名称、消息名等。紧挨关键行单元之后的行单元一般是被调用函数的返回值或消息参数等, 再后面是一些表示其他信息的行单元。状态机通过日志行的关键行单元来判断是否对其敏感。对敏感日志行, 通过判断函数的返回值或消息参数以及其他信息决定如何进行状态转移。出租客户端数据采集过程的日志片断如下:

```
...
RecRead - 1
ShowMessage
ReadAgain 1
RecRead 1
RecCheck 1 200060 2004 2 1 14:49:20 15:27:25 37.7 3 588.2 85
RecFinish - 1
RecRead 1
RecCheck 1 2000060 2004 2 1 18:43:02 19:08:07 28.5 2 13.6 64
RecFinish - 1
RecRead 1
RecCheck 1 2000060 2004 2 2 9:51:17 9:52:43 .6 0 14.4 10
...
RecFinish 1
RecsCheck 1
...
```

2.3　测试状态机与状态机定义语言

有限状态机 FSM(Finite State Machine, 以下简称状态机) [4] 被广泛应用于诸多领域, 如电路设计[5]、通信协议测试[6] 等。普通的状态机所提供的描述能力有限, 用来描述软件的执行过程能力尚显不足。结合文献[3] 中所使用的状态机和文献[7] 中提出的 PSM(Protocol State Machine) , OLFTS 中定义了一种新的状态机——测试状态机 TSM(Testing State Machine) 。

定义 1　TSM。一个 TSM 为一个 11 元组:

$TSM = <n, S, s_0, S_F, I, O, V, P, A, C, T>$, 其中, n 表示状态机的名称; $S = \{ s_1, s_2, \cdots, s_l \}$ 表示有限的状态集合; $s_0 \in S$ 表示状态机的初态; $S_F \subseteq S$ 表示状态机的终态集合; $I = \{ i_1, i_2, \cdots, i_m \}$ 表示有限的输入集合, OLFTS 中为日志行; $O = \{ o_1, o_2, \cdots, o_n \}$ 表示有限的输出集合, OLFTS 中为状态机向外输出的统计信息等; $V = \{ v_1, v_2, \cdots, v_i \}$ 表示状态机内有限的变量集合; 变量 v_j 的值域记为 $D(v_j)$, 则 $P = \{ p_i \mid p_i : \prod D(v_j) \rightarrow (True, False), v_j \in V \}$ 表示有限谓词集; $A = \{ a_i \mid a_i : \prod D(v_j) \rightarrow \prod D(v_k), v_i \in V, v_k \in V \}$ 表示有限的动作集合; $C: I \times P \rightarrow (True)$ 表示创建条件; $T: S \times I \times P \rightarrow S \times A \times O$ 表示转移函数, 对于任意 $t \in T$, $t = (s_j, j, p, s_k, a, o)$, 表示 TSM 在状态 s_j 下如果有输入 j 并对谓词 p 进行测试结果为真, 则转移到状态 s_k, 执行动作 a, 并作输出 o。

与普通状态机相比, TSM 增加了谓词集和动作集, 采用了更严格的转移函数, 描述能力更强, 比普通状态机更适合描述软件的执行过程、应用于软件测试。

虽然可采用普通的计算机语言实现 TSM, 但为了方便 TSM 的设计, OLFTS 中开发了一种支持层次化设计和自定义函数的状态机定义语言 TSML。其 EBNF[8] 描述如下所示:

```
<TSM> ::= <Precode> { <InnerStateMachine> } <StateMachine>
<Precode> ::= [ '{' <NativeCode> '}' ]
<InnerStateMachine> ::= innerTSM <InnerTSMName>
  '{' <InitState> <FinalStates> <Transit> { <Transit> } '}'
<StateMachine> ::= TSM <TSMName> '{' <UtilCode>
  <Creation> <InitState> <FinalStates> <Transit> { <Transit> } '}'
<InnerTSMName> ::= <Identifier>
<InitState> ::= initState = <State> ;
<FinalStates> ::= finalStates = '(' <State> { , <State> } ')' ;
<Transit> ::= on <State> key <Key> <Condition> to <Destination> <Do> ;
<TSMName> ::= <Identifier>
<UtilCode> ::= [ '{' <NativeCode> '}' ]
<Creation> ::= create when <Key> <Condition> <Do> ;
<Identifier> ::= <Letter> { <Letter> | <Number> }
<State> ::= =: <Identifier>
<Key> ::= # <Identifier>
<Condition> ::= [ condition <BoolExp> ]
<Destination> ::= <State> | <InnerTSMName>
<Do> ::= [ do <ActionCode> ]
<BoolExp> ::= '{' <NativeCode> '}'
<ActionCode> ::= '{' <NativeCode> '}'
<Letter> ::= a | b | … | X | Y | Z
<Number> ::= 0 | 1 | … | 8 | 9
```

以状态机代码用 Java 语言实现为例, 其中, <Precode> 为 Package, Import 等语句, 编译后出现在状态机代码的头部; <NativeCode> 为 Java 代码, 将被原样编译到状态机代码中相应的部位; <UtilCode> 为自定义属性、方法的 Java 代码, 编译后将出现在状态机代码中状态机类的属性、方法定义部分; <BoolExp> 为 Java 的布尔表达式, 实现是否创建状态机、是否执行状态转移等的判断; <ActionCode> 为访问属性、调用方法以及输出信息等的 Java 代码, 负责实现状态转移后的动作。

为方便使用, TSML 中预定义了一些全局变量和函数, 可在 <NativeCode> 部分直接使用, 例如 CL 表示当前日志行, 方法 CL. getInt(n) 以整数形式返回当前日志行的第 n 个行单元。

一个由 TSML 定义的 TSM 可以包括一个主状态机和若干内部状态机。编译后, 在状态机代码中, 内部状态机被编译到所属的主状态机中, 作为其一部分, 体现转移关系。主状态机具有前置代码、名称、辅助代码、创建条件、初态、终态集和转移定义。由于内部状态机作为主状态机一部分, 所以只具有名

称、初态、终态集和转移定义。TSML 通过 <InnerStateMachine>, <UtilCode>, <ActionCode>等支持了构建内部状态机的层次化设计方法,提供了定义并访问自定义属性、函数的功能。由于采用 TSML,扩展日志分析法简化了测试状态机的构造,增强了构造大型测试系统和测试复杂系统的能力。

2.4 状态机容器与状态机代码

状态机容器包括一个状态机创建器、状态机数组以及其他辅助部分。状态机创建器负责在符合条件时创建状态机,并放入状态机数组中。状态机编译器编译输出的状态机代码采用 State 模式构建[9]。每一个 TSM 编译后得到一个状态机类和若干状态类。状态机类继承自抽象状态机父类,负责保存状态对象、调用当前状态的转移方法等。状态机对象在建立时创建拥有的所有状态对象,放入状态数组中保存,并设置当前状态对象为初始状态对象。状态类继承自抽象状态父类,与 TSM 中定义的状态相对应,数量相等。每个状态类负责该状态下的转移判断、转移和动作执行。内部状态机在编译后成为所属状态机的若干状态类,不单独存在。

状态机容器和状态机的 UML 类图如图 2 所示。

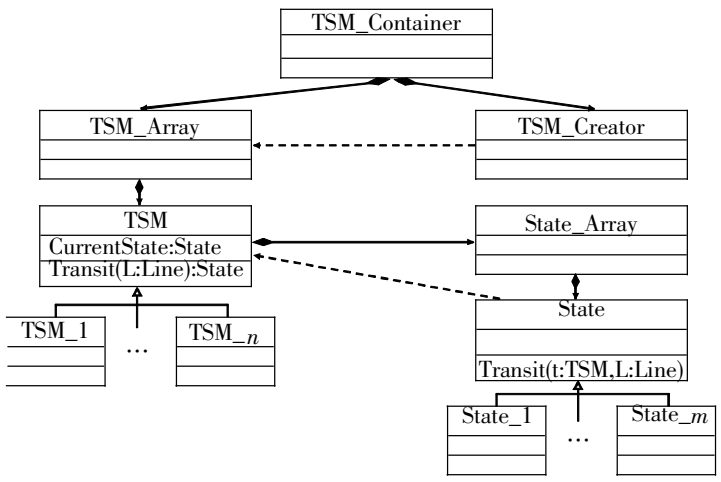

图 2 状态机容器与状态机的类图

3 应用实例

出租客户端数据采集模块主要是实现将司机卡中的运营记录采集到客户端中的功能。由于采集过程较复杂、状态多,所以该模块适宜拿出来作 OLFTS 的应用实例。

依据出租客户端数据采集模块的需求,可得数据采集基本过程如下:读取并检查司机卡基础信息。逐条读取运营记录、校验是否正确以及检查消费金额是否合法,读取完毕再总体校验一次。若所有操作均正确,则读取完毕;若过程中出现错误,则根据用户选择进行重读或宣布失败。依据以上需求及该模块的实现,可得采集过程的状态转换图,如图 3 所示。

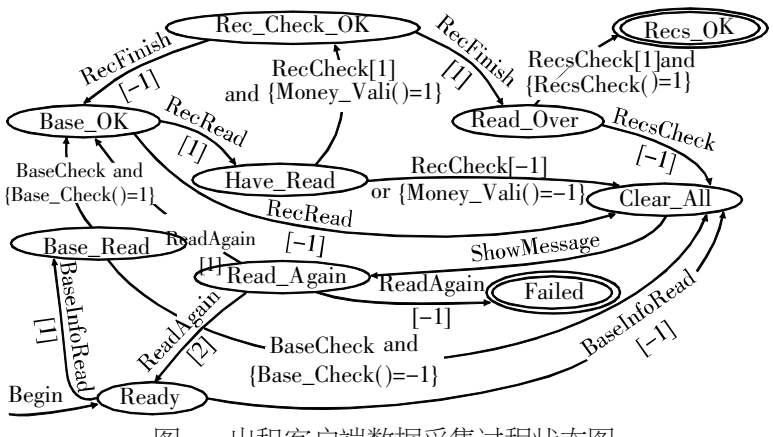

图 3 出租客户端数据采集过程状态图

由于数据校验和检查金额两种功能对判断数据的正确性至关重要,所以不仅需要测试采集模块是否按其返回值运行,还需要测试它们的功能是否正确。TSML 支持层次化状态机、具有执行自定义函数功能的特点在这时就显得非常有意义:既可以设计专门测试数据校验和检查金额功能的子状态机,连接到主状态机中实现测试;又可以在主状态机中采用状态机代码语言(Java 语言)实现数据校验和检查金额函数,在状态机进行相关转移时执行并以其执行结果作为转移判断的依据。本实例中采用后者。TaxiRecRead 的 TSML 定义如下所示:

```
{ package TaxiRecRead;
import TSM;}
TSM TaxiRecRead {
{
//Base_Check() 方法的定义
//Money_Vali() 方法的定义
}
create when #Begin;
initState = :Readey;
finalStates = (:Failed,:Recs_OK);
on: Ready key #BaseInfoRead condition{CL.getInt(1) ==1}
   to: Base_Read;
on: Ready key #BaseInfoRead condition{CL.getInt(1) == -1}
   to: Clear_All;
on: Base_Read key #BaseCheck condition
     {Base_Check(CL) == -1} to: Clear_All;
...
on: Read_Again key #Read_Again condition
   {CL.getInt(2) == -1} to: Failed;
}
```

4 结束语

本文提出了一种扩展日志分析法,介绍了该方法的实现系统 OLFTS 的结构、采用的测试状态机 TSM 和 TSM 的定义语言 TSML。TSM 比普通状态机描述能力更强,TSML 支持层次化状态机、具有执行自定义函数的功能。由于采用了 TSM 和 TSML,扩展日志分析法具有使用简单、描述能力强等特点,较适合用于关联复杂的大型系统的测试。该方法已在一卡通系统的测试中取得了良好的效果。

参考文献:

[1] Hartmann, D J Robson. Approaches to Regression Testing[C]. Proc. of Int'l Conf. Software Maintenance, Phoenix, AZ, US: IEEE Computer Society Press, 1988. 368-372.

[2] D S Roseblum. A Practical Approach to Programming with Assertions [J]. IEEE Trans. Software Eng., 1995, 21(1):19-31.

[3] J H Andrews, Y Zhang. General Test Result Checking with Log File Analysis[J]. IEEE Trans. Software Eng., 2003, 29(7):634-648.

[4] D Thomas, A Hunt. State Machines[J]. IEEE Software, 2002, 19(6):10-12.

[5] C Pixley. A Computational Theory and Implementation of Sequential Hardware Equivalence[J]. Computer Aided Verification, 1990: 293-320.

[6] B Sarikaya, G Bochmann. Synchronization and Specification Issues in Protocol Testing[J]. IEEE Trans. Comm., 1984, 32(14):389-395.

[7] 张有弢, 杨培根, 等. TCS: TCP 协议一致性测试系统的设计和实现[J]. 电子学报, 1998, 26(5):106-108.

[8] 吕映芝, 等. 编译原理[M]. 北京: 清华大学出版社, 1998.

[9] E Gamma, R Helm, *et al*. Design Patterns[M]. Boston, MA, USA: Addison-Wesley Professional, 1995. 305-315.

作者简介:

齐林(1978-),男,河北廊坊人,硕士研究生,主要研究方向为软件工程、软件测试等;郭陟(1974-),男,浙江湖州人,博士,主要研究方向为软件体系结构、系统安全;顾明(1962-),女,辽宁沈阳人,清华大学软件学院信息系统与工程研究所所长,副教授,主要研究方向为分布式应用支撑平台、系统安全等。