基于 XML 封装关键字的 GUI 自动化测试系统①

胡 骥

(航天信息股份有限公司, 北京 100195)

摘 要: 研究如何提升 Web 应用程序自动测试效率的问题. 利用 XML 技术设计并封装关键字, 完成自动化测试脚本的编写, 自动测试引擎解析 XML 文件, 解释关键字, 导航测试执行并生成报告. 以 ERP 产品的应用案例表明该系统达到了测试脚本分层设计, 提升可读性和可维护性, 降低编写难度, 提升效率的目的.

关键词: XML; 关键字; Web 应用; 自动化测试

GUI Automatic Test System Based on Keywords in XML Technology

HU Ji

(Aisino Corp, Beijing 100195, China)

Abstract: The GUI automatic test system based on keywords in XML technology is proposed to research on how to improve Web application automatic test efficiency. Using XML technology to design and package keyword, finish writing automated test scripts, test engine to parse the XML file, the explanation key navigation test execution, and report generation. In the case of ERP products show that the system can achieve the test script design, enhance the readability and maintainability, reduce the difficulty of preparation, and enhance the efficiency of the testing procedure.

Key words: XML; keywords; Web application; test automation

随着计算机软件技术和互联网的发展, Web 应用程序得到了越来越多的应用. 传统的 Web 应用 GUI 自动测试方法, 通常用脚本语言编程形成测试脚本, 实现测试逻辑, 驱动被测应用, 实现自动测试的目的[1]. 许多测试工具如 HP 公司 QTP、开源测试工具 Selenium 等都是这样的原理. 商业工具通常提供录制的功能, 可以记录测试人员的操作, 产生测试脚本.

这样的方法的弊端在于: 采用录制的方法, 产生的脚本是硬编码的, 非结构化的, 很难维护. 如果用编程的方法, 对测试人员的编程能力要求较高. 为此, 传统方法较好的工作模式是: 测试人员设计测试用例, 形成测试用例文档, 自动测试开发人员实现测试脚本, 实际上是将测试用例翻译成脚本语言, 调试运行并维护脚本. 这样的工作模式带来的问题是: 人员之间的交互可能造成理解歧义和不能同步, 增加了人力资源的需求和沟通成本. 普通的软件公司不可能提供大量的人员成本和时间, 迫切需要一种轻量级的、使用更方便、更利于维护、更省时省力的方法应用于系统级自动测试.

为了克服传统自动化测试方法上的不足,人们做了很多探索和改进, 研制了各种测试框架. 其中, 目前应用比较广泛的主要有两类: 第一类属于数据驱动测试框架(The Data-Driven Testing Framework),将测试数据从测试脚本中分离出来, 形成数据文件, 驱动测试执行的流向[2]. 另一类比较流行的是关键字驱动框架(The Keyword-Driven Testing Framework),进一步将关键字封装的业务逻辑从测试脚本中抽象出来, 从而减少维护工作量[3]. 但是, 这种情况下, 测试人员仍然需要学习脚本编程. 而且, 关键字封装的业务逻辑往往以二维表的方式实现, 层次不丰富, 难以实现复杂的测试流程控制, 其正确性也难以校验.

为此, 本文提出一种方法, 并基于此方法构建了一

① 收稿时间:2014-06-01;收到修改稿时间:2014-07-31

套自动测试系统. 实现了测试驱动、测试脚本、测试数据的分离, 使得开发和测试人员各司其职, 测试人员专注于业务逻辑, 无需编程, 提高了测试脚本的可读性和可维护性. 由于脚本可读性的提高, 测试人员不必再编写一份测试用例, 测试脚本相当于一份可自动执行的测试用例. 可维护性体现在: 如果应用程序的用户界面发生了变化, 一般情况下, 修改自动测试引擎的测试执行导航模块即可, 测试脚本无需修改.

1 系统工作原理

可扩展标记语言(XML)是一种具有结构性的标记语言, 是 W3C 的推荐标准[4]. 其简单易用, 扩展性强, 使 XML 成为数据交换的一种理想的公共语言. 通过定义 XSD Schema 文件, 可以对 XML 文件的正确性进行验证[5].

利用这个特点可以定义一整套 XML 元素及其类型、属性、约束、数据、枚举值, 用以描述含义丰富的测试关键字. 测试用例设计人员按照关键字构造 XML 文件, 构造测试用例逻辑和测试数据, 组合成不同的测试用例集. 测试平台开发人员开发自动测试引擎, 自动测试引擎解析 XML 文件, 解释关键字, 导航测试执行. 这样, 组成一套完整的 GUI 自动化测试系统, 包括关键字描述模块、测试用例模块、测试报告格式模块、GUI 对象库(可选)和自动测试引擎.

本系统定义的关键字和应用系统的具体业务无关, 因此具有通用性. 只需对自动测试引擎的测试执行导航模块稍加修改即可应用于不同的系统测试.

2 系统设计实现

2.1 关键字描述模块

关键字描述模块描述测试关键字, 用以定义测试用例的组织结构、框架、组、测试流程流转、描述操作、定义验证点、设置测试数据存取方式.

以 XML 架构文件(*.XSD)形式实现基本结构, 如图 1 所示, 主要包括:

测试框架类: 测试用例集、测试用例、测试用例前置条件、后置条件, 用于定义测试用例的组织结构;

引用类: 外部测试用例集、测试数据表, 用于提取公用测试用例和引入测试数据;

变量类: 变量、结构体, 用于参数化测试数据;

操作类: 登录系统、菜单、工具栏、输入域、查询、运行 SQL 语句、运行程序、暂停, 用于执行测试步骤;

输入域类: 文字、选择、表格、用户提示、按钮、页签、树、条目, 用于输入域操作;

验证类: 界面元素值、界面元素属性、对话框、数据库、文件, 用于设置各类验证点;

验证逻辑类: 等于、不等于、包含、不包含、大于、小于、大于等于、小于等于, 用于设置验证逻辑;

组合类: 组合输入域、网页对话框, 用于控制测试流程流转;

数据驱动类: 定步长数据驱动、变步长数据驱动, 用于设置测试驱动.

图 1 关键字定义文件结构图

通过定义关键字, 并以其为基础构建测试用例, 便可实现将测试逻辑从测试脚本中抽取出来. 如果需

要新增关键字, 则修改架构文件并在自动测试引擎解析模块中添加相应部分即可. 架构文件还可以起到校验作用, 能够保证测试用例结构的正确性.

2.2 测试用例模块

测试用例模块中存储多个测试用例, 为 XML 格式文件, 包括测试组织结构、流程控制、操作、验证点、测试数据存取方式等. 如下所示, 为某 ERP 产品测试用例示例.

```
<!--*****************批量录入***************-->
<testcase id="1" name="核算" checkpoint="数据库">
 <operation type="菜单">财务/凭证录入</operation>
  <group type="数据驱动">
    <operation type="输入">
     <target type="文字" name="凭证号">凭证:批量合法:列:凭证号</target>
    </operation>
    <operation type="输入">
     <target type="表格" identify="column:序号">凭证:批量合法:列:摘要</target>
    </operation>
    <validate type="界面">
     <target type="表格" identify="column:汇率">凭证:批量合法:列:序号~贷方</target>
    </validate>
  </group>
 <operation type="工具栏">保存</operation>
</testcase>
```

在此示例中, 定义了测试用例的输入域、操作流程、验证点及数据驱动等, 组成了一个完整的测试用例逻辑. 由此可见, 测试用例结构清晰, 逻辑性和可读性良好, 配合注释可以很好的代替测试用例文档, 而这文档又是可以自动执行的.

为实现良好的可读性, 和传统方法不同, 在本系统中, 对于输入域的元素采用输入域元素之前的标签文本而非元素本身来定位元素.

2.3 测试报告模块

测试报告是 XML 格式的, 测试报告格式模块中存储多个 XSL 文件, 用来定义测试报告显示格式. 即测试报告格式和数据是分离的.

2.4 GUI 对象库(可选)

与传统系统不同, GUI 对象库在本系统中不是必需的. 本系统在运行时搜索 HTML DOM Document 来定位元素对象. 因此, GUI 对象库主要存储 GUI 元素关键字描述和 ID 的映射关系, 起到加速定位的作用. 而不是必需的, 也不需要任何人工维护.

2.5 自动测试引擎

自动测试引擎如图 2 所示, 包括: 项目管理模块、解析模块、导航模块、GUI 元素定位模块、GUI 元素处理模块、消息处理模块、验证模块、日志模块、通用数据模块, 以及多线程和 Win32 API 模块.

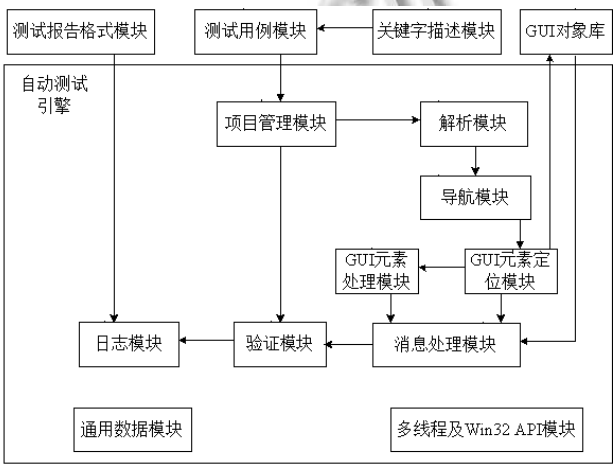

图 2 自动测试引擎结构图

项目管理模块可建立测试项目、添加测试用例、设置测试运行时参数, 配置选项; 当开始执行测试后, 调用解析模块逐一解析测试用例集中的测试用例;

解析模块利用反串行化技术解析测试用例 XML 文件, 保存到数据结构类中;

导航模块执行各测试用例集、测试用例、递归执行引用的测试用例, 解释各类关键字, 调用 GUI 元素定位模块定位 GUI 元素;

GUI 元素定位模块根据对象描述, 找到相应的 GUI 元素, 按照操作描述, 调用消息处理模块. 在找到相应元素后, 如元素具有 ID 值, 即自动建立缓存至 GUI 对象库, 在下一次执行即可通过元素 ID 定位元素, 从而加速测试执行.

GUI 元素处理模块处理各种复杂的 Web 页面 GUI 元素, 如表格、翻页、参照输入等, 按照关键字的描述, 调用消息处理模块, 模拟各种输入操作;

消息处理模块在元素对象上触发各种消息, 以模拟各种操作;

验证模块执行各种测试验证, 方式包括界面元素

值、界面元素属性、对话框、数据库字段值、文件;

日志模块记录测试结果, 将测试过程中发生错误的调试信息记录到测试日志, 将每一步的测试结果、测试统计信息输出到 XML 格式的测试报告, 调用测试报告格式模块, 显示格式化的测试报告;

通用数据模块处理各种文件输入输出、数据库操作, 以及定义各种公共数据结构, 做数据交换;

多线程和 Win32API 模块提供多线程处理和调用操作系统 Win32API 功能. 通常, 网页对话框、消息提示对话框等模式对话框的出现会暂停主线程的执行, 等待人工处理. 本系统通过启动多线程, 采用轮询方式检测对话框的产生, 如果存在将记录对话框的信息, 关闭对话框, 然后回到主线程继续执行. 这也使得本系统具有处理非预期出现的对话框的能力.

3 系统工作流程

图 3 显示了本文提出的基于 XML 封装关键字的 GUI 自动化测试系统的工作流程. 其具体步骤如下:

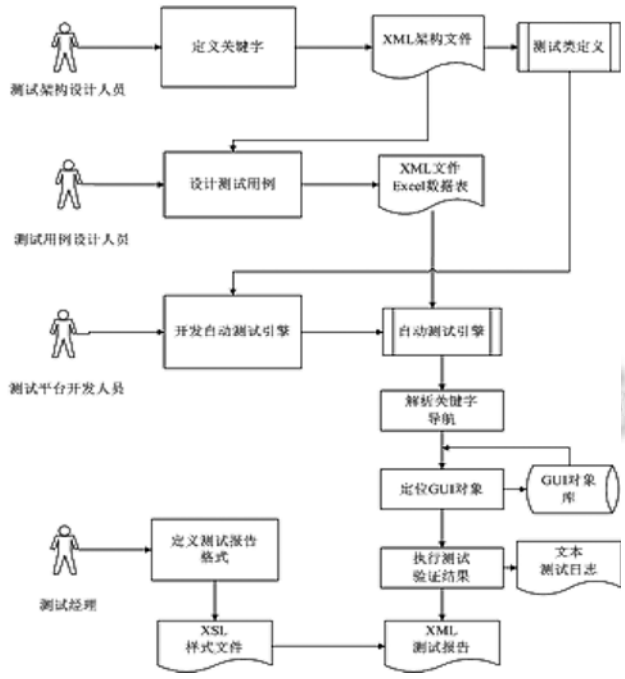

图 3 工作流程图

步骤 1: 测试架构设计人员定义一整套 XML 元素及其类型、属性、约束、数据、枚举值, 用以描述测试关键字, 保存为 XML 架构文件, 通过 XML 转换工具, 可通过 XML 架构文件自动生成各测试类的定义, 提供给自动测试引擎开发人员使用;

步骤 2: 测试用例设计人员按照关键字构造 XML 文件, 构造测试用例逻辑和测试数据, 组合成不同的测试用例集, 可以编写公用的测试用例, 供其他测试用例设计人员引用. 测试数据和预期结果值在元素值中定义, 既可用常数表示, 也可用变量表示, 也可用引用数据文件表示. 数据文件为 Excel 格式;

步骤 3: 测试平台开发人员开发自动测试引擎, 解析 XML 文件, 解释关键字, 导航测试执行, 定位 GUI 元素, 执行测试并验证结果, 产生文本测试日志和 XML 格式测试报告.

4 应用案例

基于本文所述方法研发的自动化测试系统, 在某大型 ERP 产品得到了应用. 众所周知, ERP 产品具有规模大、业务规则复杂、功能实现多样的特点. 本文所述 ERP 产品, 包括财务、供应链、生产制造、CRM、OA 等 12 个子系统, 采用 J2EE 架构开发, 代码规模在 50 万行以上. 采用本系统进行自动化测试. 该产品的测试情况如表 1 所示.

表 1 测试情况统计

	测试用例数量	设计测试用例工时(人日)	执行 3 次工时(人日)	研发测试系统工时(人日)
手工测试用例	10,642	532	1596	
自动测试用例	11,558	1537	6	390
合计	22,200			

续表 1 测试情况统计

	服务器一年折旧费用(元)	费用总计 (工时按 600 元/人日计算) (元)
手工测试用例		1,276,800
自动测试用例	20000	1,179,800
合计		2,456,600

在该项目中, 自动测试用例数量为 11558 个, 占全部测试用例数量比例大约为 52%. 如果此部分测试用例全部采用手工测试, 则需要 2312 人日, 费用大约为 1,387,200 元. 由此可见, 采用本系统执行自动测试, 在执行三轮测试的情况下, 费用相比手工测试节省了 207,400 元, 收益率大约为 14.95%. 通过计算可知, 如

扣除研发测试系统的费用, 采用本系统执行自动测试, 大约执行两轮测试即可实现正收益.

相比传统方法, 采用传统自动测试系统, 根据有关案例测算[6], 一般需要经过四轮左右的测试, 才可能实现收回自动化投入的成本.

5 结语

本文介绍了一种基于 XML 封装关键字的 GUI 自动化测试系统. 该系统具有通用性, 只要稍加修改, 在各种被测应用系统均可应用. 通过应用该系统, 实现了测试脚本的分层设计, 提高了测试脚本的可读性和可维护性, 降低了脚本编写的难度. 解决现有脚本化自动测试方法中将测试驱动、测试逻辑、测试数据混杂在一起, 测试脚本无法直接用于测试用例评审, 必须同时维护一份测试用例和一份测试脚本的技术问题. 同时, 使各类人员分工协作, 并行工作, 各展所长.

参考文献

1 Nguyen HQ.冯学民,等译.Web 应用测试.北京:电子工业出版社,2003.

2 Fewster M, Graham D.舒智勇,等译.软件测试自动化技术与实例详解.北京:电子工业出版社,2000.

3 柳胜.软件自动化测试框架设计与实践.北京:人民邮电出版社,2009.

4 W3C. Extensible Markup Language (XML). http://www. w3. org/XML/. 2013.

5 McCaffrey JD.刘晓伟译. .NET 软件测试自动化之道.北京:电子工业出版社,2007.

6 Graham D, Fewster M.朱少民等译.自动化测试最佳实践.北京:机械工业出版社,2013.