第32卷 第20期 计 算 机 工 程 2006年10月
Vol.32 № 20 Computer Engineering October 2006
·软件技术与数据库· 文章编号：1000—3428(2006)20—0078—03 文献标识码：A 中图分类号：TP311

面向构件化软件的合约检查测试框架

张毅坤，叶 涛，邢传玉
(西安理工大学计算机科学与工程学院，西安 710048)

摘 要：基于构件软件开发的主要思想是使用现存的构件来建构软件系统。而这样的系统由于构件本身的特点导致了许多测试困难。B. Meyer将构件与其客户代码之间的关系形式化地定义为一种合约，它严格限定了构件对象之间的交互规则。通过对合约的监视和检查，可以容易地发现构件之间的交互错误，从而达到集成测试构件化软件的目的。该文提出了一种基于合约检查的构件集成测试框架(contract-checking test framework，CCTF)。讨论了该框架合约检查的思想、5大功能模块以及其测试流程，并介绍了将CCTF应用到构件化软件测试平台实现的一些关键技术。

关键词：构件；合约检查；CCTF；构件化软件集成测试

Contract-checking Test Framework for Component-based Software

ZHANG Yikun, YE Tao, XING Chuanyu
(School of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048)

【Abstract】 The main idea of component-based software development (CBSD) is to use existing components for building up software systems. However the features of components result in a great number of difficulties in testing such systems. B. Meyer defines the relationship of component and client code as a formal contract, which firmly restricts the rule of interaction among component objects. Monitoring and checking a contract can easily find the interaction errors among a pair of components. This paper proposes a component integration test framework based on contract checking, contract-checking test framework(CCTF). And discuss the principle of contract checking, five function modules, testing process, and some relevant crucial techniques in applying CCTF to implement components-based software test platform.

【Key words】 Component; Contract-checking; CCTF; Components-based software integration test

B.Mayer首次提出的合约化设计(Design By Contract，DBC)[1]理论，不仅仅给构件化软件提供了一种新的设计方法，同时也给测试构件化系统提供了新的测试思路。本文在研究合约化设计理论的基础上，结合文献[2~5]提出的多种构件化软件测试策略，提出了一种新的测试工具框架，充分考虑了文献[4]提到的构件化软件的特点，利用框架引入的构件接口合约文档来描述系统中构件的行为，通过解析和分析该文档检查构件在系统运行中的交互行为，从而达到集成测试构件化软件的目的。

1 合约检查的思想

1.1 合约的定义

B.Mayer[1]在一文中将构件与其客户代码之间的关系形式化地定义为合约(Contract)。合约是管理对象之间交互的一组规则。常见的合约元素包括：前置条件，后置条件，不变式等。合约表明了过程调用方(客户方)与实现方(服务方)相互的职责和权利。调用方只有在满足前置条件的情况下，才能调用对应的过程；实现方承诺当过程结束时，后置条件中指明的工作将被完成，且不变式仍然满足。

因此，通过合约可以容易地区分软件失效时的责任方：如果前置条件被违反,则应该在调用方寻找错误；如果后置条件或不变式被违反，那么责任在实现方。合约概念的引入有助于减少冗余的检查代码，提高软件设计的效率和运行的性能，更重要的是便于开发出自动检查合约的测试工具平台，准确地定位错误和故障位置，提高构件集成方测试系统的效率。

B.Mayer同时指出合约化软件的构建可分为两个阶段[1]：合约的定义以及合约的实现阶段。合约定义阶段是最重要的阶段。该阶段关注于接口合约的定义以及构件的可复用性和互操作性，它属于软件体系结构的范畴。一个系统被描述为构件以及连接件的组合，构件之间的连接是系统设计需要重点考虑的问题。这个方面影响了整个系统的复用性，可替换性，以及系统的可升级性。

合约实现阶段主要关注对“粘贴”代码的开发。这些代码是不同构件之间交互的“通道”。在一个或者多个候选构件被确定后，需要评价其中的每一个构件与系统已经选定的构件集成时交互行为。很明显在这一阶段，测试扮演了相当重要的角色。事实上测试者根据事先定义的合约可以容易地开发出有效的测试以衡量候选构件。

1.2 合约检查的思想

鉴于以上的讨论，本文提出了合约检查的测试策略。图1描述了构件、应用程序(客户代码)，以及合约检查器之间的关系，展示了合约检查的基本思想。从图1中可以看出应用程序与构件的交互被自动生成的合约检查器所代理。应用程序在调用构件的每一个方法之前和之后都需要经过检查器对不同合约元素进行的必要检查。具体来说，就是调用方在调

基金项目：陕西省自然科学基金资助项目(2001x20)；陕西省教育厅科研基金资助项目(00JK265)
作者简介：张毅坤(1958－)，男，教授，主研方向：软件工程；叶 涛、邢传玉，硕士生
收稿日期：2006-01-12 **E-mail：**Ykzhang@xaut.edu.cn

用构件提供的服务之前，合约检查器检查合约的前置条件元素是否满足，如果满足，就执行真正的构件服务方法；如果不满足，就抛出自定义的异常，记录结果，执行下一个测试用例。在构件方法调用完毕后合约检查器还要对返回结果进行后置条件元素的检查。同样，如果满足条件，说明本次调用已正确完成，记录结果，准备进行下一个测试；如果不满足，抛出自定义异常，记录测试结果。

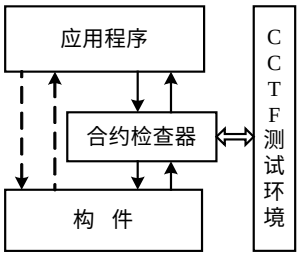

图 1 合约检查原理

通过在构件与应用程序之间插入合约检查器，使构件与应用程序实现隔离，有效地提高了构件与应用程序交互通信的可观测性，构件行为的可追踪性，极大地减少了定位错误的时间，是一种可行的构件化软件测试策略。

2 合约检查测试框架 CCTF

2.1 CCTF 概述

为了实现合约检查的思想，框架的设计目标是针对构件化软件的集成测试。通过使用该框架使构件使用者可以测试已经集成到实际系统中的单个或者多个构件，检查构件与系统其他部分的交互与通信，记录各种通信消息和异常。框架设计包含以下 5 个主要模块：

(1)CCTF 环境模块(CCTF-Environment Module)：是一个核心模块，它负责初始化其他的各个框架模块，控制着整个测试过程的进行；

(2)配置模块(Configuration Module)：主要解析测试用例文档，完成可执行测试用例的构造；

(3)合约检查器产生器模块(ContractChecker-Generator Module)：是框架重点设计模块，该模块解析规范化的接口合约文档，产生针对构件合约的检查代码；

(4)测试驱动产生模块(TestDriverGenerator Module)：依据配置器产生的可执行测试用例的要求，完成构件的初始化，构件与环境，构件与合约检查器的连接，产生完整的测试驱动代码；

(5)测试报告模块(TestReportGenerator Module)：在测试驱动运行时提取测试结果并产生多种形式的测试报告，供测试者查阅。

图 2 描述了框架的 5 大模块以及模块与模块之间的关系。图 2 中除了显示了 5 大模块之外，还标识出支持框架运行还需要的 4 种 XML 输入文档，分别为：

(1)测试配置文件(Testconfig File)：该文件具体定义了需要运行的测试用例以及执行测试用例的顺序；

(2)构件描述(Descriptions of component)：该文件同样使用可扩展标记语言 XML 描述构件本身信息,主要信息包含了被描述构件各个接口方法与哪些构件或是环境发生了 交互；

(3)构件合约(Contract of component)：该文档是框架的核心，它是对构件对象通信规则的定义，是框架合约检查器的输入文档。该文档具体规定了构件每一个接口方法的 3 种合约元素(前置条件，后置条件，不变式)。这些构件合约放在合约库中统一管理；

(4)结果存储(ResultStore)：框架设计使用文件形式保存测试结果和日志文件。该文档包含了每一个测试用例运行结果、测试结果、错误位置(如果发生错误的话)，以及接口测试充分性等信息。

框架模块与模块、模块与输入文档之间的关系在图 2 上使用类似于消息的表示方式予以说明。

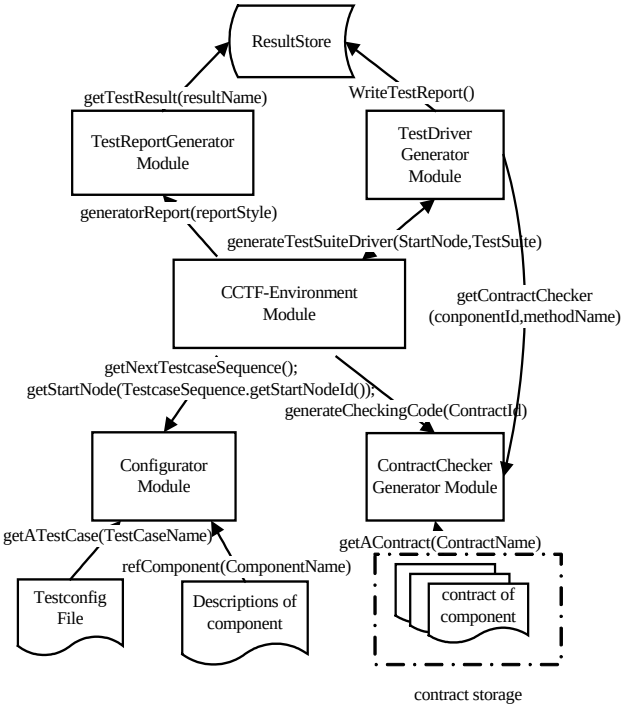

图 2 CCTF 测试框架

2.2 CCTF 的测试流程

测试的流程分为 3 个步骤：(1)分析测试配置文档，产生可执行测试用例，完成合约检查器的构造。(2)测试驱动产生器根据测试用例连接构件和合约检查器。(3)编译和运行测试驱动，产生测试结果，输出测试报告。每一步的任务如下：

(1)在已经给定测试配置文档和测试用例文档的情况下，测试环境启动测试配置模块，后者调用 XML 解析器分析测试配置文档，然后根据该文档中的<TestSuite>标签找到第 1 个测试集。测试环境获取该测试集的入口构件也就是起始构件。配置模块要求环境给出该起始节点的合约检查器，根据入口构件描述文档确定与该构件相关的其他构件，向合约检查产生器导出这些构件 Id。如果测试环境的“产生合约检查”配置项被设置为真的话，合约检查产生器会向合约库请求相应构件接口的合约描述，并产生相应的合约检查器。

(2)测试环境使用带有测试集和入口构件作为参数的消息启动测试驱动产生器。驱动产生器依据测试集文档的描述连接相关的构件以及相关的合约检查器，完成整个测试驱动的自动产生功能。

(3)编译和运行第(2)步产生的测试驱动，在运行过程中测试驱动和合约检查器并向测试环境报告测试结果，同时将结果写入测试结果文档。由测试报告产生器读取这些文档，显示符合客户需求的测试报告格式。

另外，该过程描述了一个测试用例集的测试过程，在配置器读入下一个测试集时，以上 3 步重复进行。直到运行完测试配置中定义的所有测试集。

2.3 CCTF 的优点

合约检查测试框架的输入文档均采用 XML 文档格式，通用程度高，与其他已知的一些构件化软件集成测试策略相比，具有不需要学习使用新的构件接口定义形式化语言、灵活度高的优点。为了避免在构件化软件集成测试中功能测试组合爆炸的缺点，设计了专门的配置器，通过解析接口合约文档和测试配置文档可以进行动态配置、连接和初始化构件，从而完成相应的测试任务，在读取下一测试用例集之前，释放前一次初始化和连接的构件，使构件系统回到初始状态，然后完成重新配置和连接。

3 CCTF 的构件化软件测试平台原型系统技术实现

具体将测试框架应用到整个构件化软件测试平台的实现时，所涉及的内容很多。其中包括：各个功能部件的实现，测试环境的搭建，各种 XML 输入文档的描述，测试集/测试用例的驱动和管理，测试结果采集、比较和分析等。由于篇幅限制，本文将重点讨论实现框架中的以下关键技术：测试用例 XML 文档的设计，合约检查产生器的设计实现，以及测试驱动的产生技术。

3.1 测试集/测试用例文档的设计

测试集/测试用例文档可以手工产生也可通过形式化的规约(如 UML 规约等)自动生成。无论采用何种产生手段，都要符合本框架定义的、使用 XML 形式描述的测试用例规范，这样才能使用框架完成测试过程。

对于一个构件化软件来说，测试用例可以成千上万，通常使用层次来组织和管理测试用例。我们的测试平台原型系统将测试用例分为 3 个层次：测试配置层，测试集层和测试用例层。每一个层次代表了一种类型的 XML 文档。三者的关系如图 3 所示。

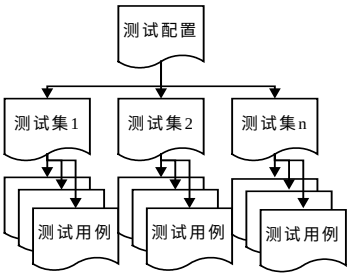

图 3 测试用例的组织

测试配置层文档可以简单的包含一个或者多个<TestSuite>标签，其内容是测试集文档名称。配置层主要用来表明需要运行的若干测试集。测试集层的文档是对一个或者多个测试用例执行顺序的定义，同时指明了该测试集的入口构件，也就是首先实例化的构件。测试集文档指明该测试集的起始构件使用<startnode>标签来表示，并使用一个或多个<testcase>标签来表明需要执行的测试用例顺序，标签的内容是不同测试用例文档的名称。

测试用例层是描述测试的核心。使用以下的例子来说明。例如：测试一个 StringService 构件的 lowerTOupper 接口方法的测试用例层 XML 文档描述为：

```
<?xml version="1.0" encoding="UTF-8"?>
<testcase
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation = "TestCase.xsd">
        <testcaseId>01-0B-09</testcaseId>
      <component>
        <id>StringService</id>
       </component>
      <method>lowerTOupper</method>
      <result>
        <returntype>String</returntype>
        <equals>OTTO</equals>
      </result>
      <parameters>
          <parameter>
            <type>String</type>
            <value>otto</value>
          </parameter>
        </parameters>
      <resultchecker>
        <id>StringChecker</id>
      </resultchecker>
    </testcase>
```

例子中<component>标签指明了被测试方法的宿主构件名称。<method>指明被测试方法。<result>标签是对方法返回正确结果的定义。被测试方法输入参数数据在<parameters>标签中描述。<resultChecker>标签指明了方法返回值后置条件检查类的名称。

在原型系统中，分析了 Java 中 4 种流行的 XML 解析技术，认为 DOM4J 符合解析文档要求。一方面由于其开放源代码，更重要的是其解析 XML 文档的速度相当快，它使 CCTF 在大量测试用例条件下可以获得良好的性能特性。

3.2 合约检查产生器

合约检查产生器的输入输出边界是接口合约文档和合约检查器。

原型系统使用 XML 来定义构件接口合约，而不使用其他文献提到的形式化语言，如文献[3]中使用 Java 建模语言定义合约内容。原因一是使构件接口合约的定义与构件的实现相互隔离，保持构件的封装性；其二是使原型系统不用增加对新语言的解释编译环境支持，简化原型系统设计。

具体通过例子代码来说明合约检查产生器的工作过程。构件接口合约文档为：

```
<?xml version="1.0" encoding="UTF-8"?>
<contract
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation = "Contract.xsd">
      <interface>sample</interface>
      <invariant>x>10</invariant>
      <methodcontract>
        <methodName>setX</methodName>
          <precondition>
              <Expression>val>20</Expression>
          </precondition>
          <postcondition>
            <Expression> this'x = = this.x+val
            </Expression>
          </postcondition>
      </methodcontract>
      <methodcontract>
        <methodName>getX</methodName>
            <postcondition>
              <Expression>getX() = = this.x</Expression>
            </postcondition>
      </methodcontract>
</contract>
```

如果有一个测试用例需要对该构件的 setX()方法进行测试，那么合约检查产生器在解析到接口合约中该方法时将前置条件转化为测试驱动中的 if-else 语句，完成合约检查，如果满足则由测试驱动完成构件方法的调用，如果未能满足，向测试环境报告异常，记录测试结果。

为了尽可能的自动化整个测试过程，我们设计实现了专门的合约检查产生向导程序，负责与测试人员交互，辅助完成合约检查代码的半自动化生成工作。图 4 显示了利用合约产生向导完成接口合约文档的新建功能。 (下转第 83 页)